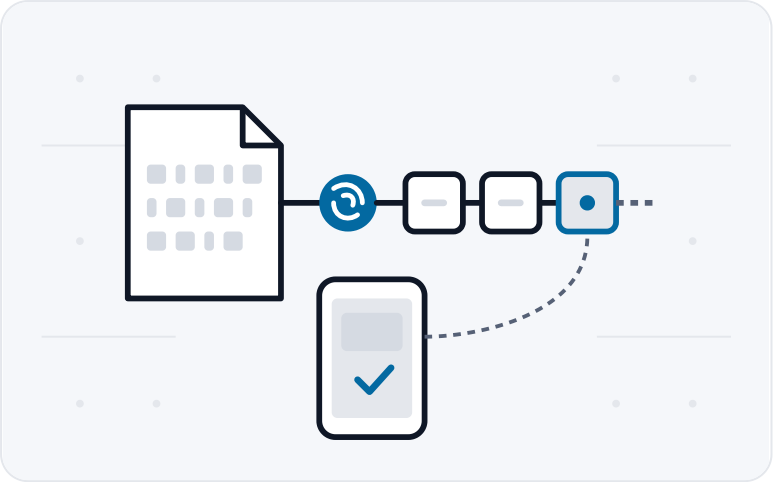
A=SIT Technology

E-Government Innovationszentrum

24. Juni 2025 Konzept · Transparenz · Digitale Identität · 32 Min. Lesezeit

Wallet Binary Transparency

Edona Fasllija



Autorinnen und Autoren	Edona Fasllija
Quelle	https://technology2.a-sit.at/veroeffentlichungen/wallet-binary-transparency/
Hinweis	Maschinell übersetzt.

Kernaussagen

- Ein öffentliches, nur anfügbares Log hält Hash, Version und Metadaten jedes Wallet-Releases fest und ergänzt so die Signatur der Entwickelnden.
- Unabhängige Rebuilder bestätigen die Reproduzierbarkeit, wenn ihr Hash mit dem Log-Eintrag übereinstimmt, und melden Abweichungen als Mismatched Build Alert.
- Ein Verifikationsmodul prüft bei der Installation Signatur, Signed Entry Timestamp und Inclusion Proof und blockiert die Installation oder warnt bei Fehlern.
- Der Entwurf umfasst fünf Phasen, von der Einreichung beim Log bis zur kontinuierlichen Überwachung durch Monitore und Prüfer.
- Für bis zu 400 Millionen Nutzende braucht das Rekor-basierte Log Skalierung durch Sharding, Replikation und statische Tile-APIs.

Binary Transparency ist eine Sicherheitspraxis, die sicherstellt, dass Software-Binärdateien überprüfbar sind und nicht manipuliert wurden. Sie zielt darauf ab, das Vertrauen in Software zu erhöhen, indem sie Mechanismen für Benutzer, Sicherheitsforscher oder Prüfer bereitstellt, um die Integrität und Authentizität einer Software- Binary zu überprüfen. Dieser Ansatz erschwert es Angreifern, böswilligen Code unter dem Deckmantel einer legitimen Anwendung zu verbreiten.

Die Transparenz von Binärdateien ist besonders nützlich, wenn es darum geht, das Vertrauen in kritische Softwaresysteme wie digitale ID-Wallets aufrechtzuerhalten, bei denen die Gewährleistung der Integrität des Codes für die Sicherheit von größter Bedeutung ist. Sie trägt dazu bei, die Risiken in der Software-Lieferkette zu mindern, und erschwert es Angreifern, unbemerkt bösartige Updates einzuschleusen oder Software zu verändern.

Bei der Anwendung von Binary Transparency auf Digital Identity Wallets speichert ein öffentliches, nur für Anhänge bestimmtes Protokoll kryptografische Hashes der Binaries und Metadaten wie Versionsnummern, Veröffentlichungsdaten und Herausgeberinformationen. Die Protokolle sind öffentlich prüfbar und gewährleisten eine historische Nachvollziehbarkeit: Änderungen an einer Binärdatei können durch Vergleich mit dem im Protokoll aufgezeichneten Hash erkannt werden. Die Benutzer können überprüfen, ob die Aktualisierungen vom rechtmäßigen Entwickler stammen und mit einer bestimmten, im Transparency Log aufgezeichneten öffentlichen Version übereinstimmen.

1. Einleitung

Systeme zur Verteilung mobiler Apps beruhen seit Langem auf der grundlegenden Annahme, dass Entwickelnde die alleinige Kontrolle über ihre Signaturschlüssel behalten. Diese kryptografischen Schlüssel dienen dazu, Anwendungs-Binaries zu signieren. Sie bieten eine Form der Authentizitätsprüfung, mit der App-Stores und Nutzende darauf vertrauen können, dass die Software aus der angegebenen Quelle stammt. Dieses Modell ist weit verbreitet, birgt jedoch grundlegende und erhebliche Schwachstellen. Signaturschlüssel können durch Phishing-Angriffe, Malware-Infektionen, mangelhaftes Schlüsselmanagement oder Datenlecks von innen kompromittiert werden. Sie lassen sich dann nutzen, um schädliche Software zu erzeugen und zu verteilen, die von legitimen Releases nicht zu unterscheiden ist. Insider-Bedrohungen stellen eine zusätzliche Gefahr dar, da jede berechtigte Entwicklerin und jeder berechtigte Entwickler mit Zugriff auf die Signaturinfrastruktur absichtlich nicht autorisierte Software oder Software mit Hintertüren veröffentlichen kann. Dieses Risiko verschärft sich, wenn Vertriebsplattformen die Verantwortung für das Schlüsselmanagement übernehmen und sich das Vertrauensmodell dadurch zu einer zentralisierten Infrastruktur verschiebt. Plattformen wie Google Play verlangen beispielsweise inzwischen, dass Entwickelnde ihre Signaturschlüssel in die Systeme der Plattform hochladen. Das vereinfacht zwar die App-Signierung und die Update-Prozesse, schafft aber auch eine Vertrauensabhängigkeit von den Sicherheitspraktiken und der Betriebsintegrität der Plattform.

Um diese Risiken zu mindern, hat die Software-Sicherheits-Community das Konzept der Binary Transparency (BT) entwickelt. Im Kern führt Binary Transparency eine überprüfbare Protokollierungsschicht in die Softwareverteilung ein. Sie erzeugt ein kryptografisch gesichertes Protokoll jedes veröffentlichten Binarys, das nur um Einträge ergänzt werden kann (append-only). Anders als herkömmliche Systeme, die sich allein auf die Sicherheit der Signaturschlüssel stützen, ermöglicht Binary Transparency es, manipulierte oder nicht autorisierte Software anhand öffentlicher Audit-Logs zu erkennen. Jedes Signaturereignis und

die zugehörigen Metadaten, etwa Binary-Hashes, Versionsnummern und Zeitstempel, werden unveränderlich in einem Transparency Log aufgezeichnet. Dieses Log basiert auf Merkle-Baum-Strukturen. Es stellt sicher, dass frühere Einträge nicht unbemerkt verändert oder entfernt werden können, und bildet eine konsistente und prüfbare Historie der Software-Releases.

Indem sie jede Version eines verteilten Binarys in diesem Log veröffentlichen, können Entwickelnde externen Prüfern, unabhängigen Rebuildern und sogar Endnutzenden ermöglichen, die Integrität und Herkunft ihrer Software zu verifizieren. Dadurch wird es deutlich schwieriger, gezielte Angriffe unbemerkt durchzuführen, bei denen schädliche Versionen nur an bestimmte Nutzende ausgeliefert werden. Außerdem verringert sich der Schaden einer Kompromittierung von Signaturschlüsseln, da schädliche Builds, die nicht ordnungsgemäß protokolliert wurden oder sich nicht aus dem Quellcode reproduzieren lassen, rasch als verdächtig auffallen. Darüber hinaus kann das Log selbst kontinuierlich von Dritten überwacht werden. Diese können die Öffentlichkeit warnen, wenn sie Inkonsistenzen, unerwartete Releases oder Versuche beobachten, die Historie der veröffentlichten Binaries umzuschreiben.

Dieses Projekt untersucht, wie sich Binary Transparency in den Entwicklungs- und Verteilungszyklus der Software der European Digital Identity Wallet (EUDI Wallet) integrieren lässt. Die EUDI Wallet wird im Rahmen des eIDAS-2.0-Rahmenwerks der EU entwickelt. Sie soll es allen Bürgerinnen und Bürgern der EU ermöglichen, offizielle digitale Nachweise wie nationale Ausweise, Führerscheine, Gesundheitsbescheinigungen und akademische Zeugnisse sicher zu speichern und vorzuzeigen. Angesichts der Sensibilität und der rechtlichen Bedeutung der von der Wallet verwalteten Daten ist es unerlässlich, dass ihre Software authentisch und nachweislich sicher ist.

Unsere Arbeit konzentriert sich darauf, mit Binary Transparency jedes Wallet-Release öffentlich prüfbar zu machen. In unserem Entwurf signieren Entwickelnde Binaries und reichen sie bei einem manipulationssicheren Log ein, das jedes Release dauerhaft festhält. Unabhängige Rebuilder können diese Binaries aus dem Quellcode reproduzieren und Reproduzierbarkeitsbestätigungen veröffentlichen. So ist sichergestellt, dass das veröffentlichte Binary dem öffentlich geprüften Code entspricht. Client-Geräte oder Installationsmodule können das Transparency Log bei der Installation abfragen, um das Vorhandensein, die Authentizität und die Integrität des Binarys zu überprüfen. Fehlt das Binary im Log, stimmt es nicht mit reproduzierbaren Builds überein oder weist es Anzeichen ungewöhnlicher Signaturaktivität auf, kann die Installation blockiert oder zur Prüfung markiert werden.

Durch die Integration von Transparenzmechanismen ersetzt unser Projekt ein Vertrauensmodell, das sich allein auf zentrale Stellen stützt, durch eines, in dem jede Person die Authentizität der Software unabhängig überprüfen kann. Damit können Regulierungsbehörden, Entwickelnde und Endnutzende selbstständig bestätigen, dass jedes Release echt und unverändert ist. Es ermöglicht außerdem Überwachung in Echtzeit und Analysen nach Vorfällen, die in einem komplexen System mit hohen Sicherheitsanforderungen wie der EUDI Wallet unverzichtbar sind. Auf diese Weise steht das Projekt im Einklang mit den Zielen von eIDAS 2.0: Es fördert einen Rahmen für digitale Identitäten, der nicht nur sicher und interoperabel, sondern auch von vornherein transparent und überprüfbar ist.

2. Hintergrund

2.1. Angriffe auf die Software-Lieferkette

Kompromittierung von Signaturschlüsseln

Signieren unter der Kontrolle der Entwickelnden ist üblich, birgt aber Risiken, die schwer zu erkennen und noch schwerer zu beheben sind. Signaturschlüssel sind in diesem System die zentralen Vertrauensanker und damit besonders schützenswerte Güter. Wenn sie kompromittiert werden, lassen sich damit schädliche Binaries erzeugen, die für Endnutzende, Update-Systeme und App-Stores völlig legitim erscheinen.

Es gibt mehrere gut dokumentierte Beispiele aus der Praxis, die zeigen, wie die Kompromittierung von Signaturschlüsseln zu großflächigen Angriffen führen kann[1], [2]. Solche Angriffe haben nicht nur große Auswirkungen, sondern werden auch immer häufiger und raffinierter. Laut dem Bericht 2025 von ReversingLabs [3] haben Angriffe auf die Software-Lieferkette weiter zugenommen, die Vorfälle sind in den vergangenen vier Jahren um mehr als 700 % gestiegen. Der Bericht hebt hervor, dass Angreifende zunehmend vertrauenswürdige Entwicklungsinfrastrukturen ins Visier nehmen, um schädlichen Code in ansonsten legitime Software einzuschleusen. Ebenso zählte die Agentur der Europäischen Union für Cybersicherheit (ENISA) Angriffe auf die Lieferkette in ihrem Bericht Threat Landscape 2024 [4] zu den sieben wichtigsten Cybersicherheitsbedrohungen, neben Ransomware, Malware und Social Engineering. Der Bericht betont, dass das Vertrauen in Signaturschlüssel allein nicht mehr ausreicht, um die Integrität von Software zu gewährleisten.

Bedrohungen der Build-Pipeline

Die Kompromittierung von Signaturschlüsseln ist ein ernstes Risiko. Eine wachsende Klasse von Angriffen zielt jedoch auf die Build-Pipeline selbst. Diese Angriffe nutzen die Lücke zwischen dem geprüften Quellcode und dem fertig kompilierten Binary aus. Selbst wenn Entwickelnde ihre Signaturschlüssel und ihren Release-Prozess sicher verwalten, kann die Build-Umgebung kompromittiert werden, um Malware oder nicht autorisiertes Verhalten in die produktive Software einzuschleusen. Statt den Quellcode zu kompromittieren oder Signaturzugangsdaten zu stehlen, konzentrieren sich Angreifende nun auf den Software-Entwicklungs- und Auslieferungsprozess und manipulieren, was gebaut und verteilt wird. Diese Angriffe zur Build-Zeit schleusen schädliches Verhalten während der Kompilierung oder Paketierung ein, oft ohne eine Spur im Quellcode oder in Versionskontrollsystemen zu hinterlassen.

Darüber hinaus verlassen sich moderne Build-Systeme häufig auf Abhängigkeiten von Dritten, die aus öffentlichen Repositories bezogen werden (z. B. Maven Central, npm, PyPI). Sind diese Abhängigkeiten nicht kryptografisch festgelegt (gepinnt) und verifiziert, können Angreifende „Dependency Confusion“ [5], [6] ausnutzen. Dabei laden sie schädliche Pakete in öffentliche Registries hoch, die dieselben Namen tragen wie interne Abhängigkeiten. Das Build-System kann diese öffentlichen Pakete fälschlicherweise auflösen, was zu unbemerkter Code-Einschleusung führt.

Eine weitere Kategorie von Risiken zur Build-Zeit betrifft die Manipulation der Werkzeuge, mit denen der Quellcode kompiliert wird. Wird ein Compiler oder ein Build-Werkzeug kompromittiert, kann es schädlichen Code in das endgültige Binary einschleusen, unabhängig davon, ob der Quellcode unbedenklich erscheint. Diese Form des Angriffs beschrieb Ken Thompson in seinem wegweisenden Vortrag „Reflections on

Trusting Trust“ aus dem Jahr 1984 [7]. Er zeigte, dass ein Compiler mit Hintertür sich unbemerkt über Generationen von Software hinweg fortpflanzen kann. Obwohl er ursprünglich als Gedankenexperiment vorgestellt wurde, gelten solche Angriffe auf die Toolchain heute als plausibel, insbesondere in Closed-Source-Ökosystemen oder unüberwachten CI/CD-Umgebungen.

2.2. Reproduzierbare Builds

Reproduzierbare Builds haben sich als grundlegende Strategie zur Verbesserung der Sicherheit der Software-Lieferkette etabliert. Sie stellen sicher, dass unabhängige Systeme bei gleichem Quellcode, gleicher Konfiguration und gleicher Umgebung identische Binaries erzeugen können. Das mag wie ein technisches Detail auf niedriger Ebene wirken, hat aber erhebliche Auswirkungen: Es erlaubt jeder Person zu überprüfen, ob ein Binary tatsächlich seinem Quellcode entspricht. Das hilft, Manipulationen, Angriffe zur Build-Zeit und andere Formen nicht autorisierter Veränderung zu erkennen.

Historisch wurde die Softwarekompilierung als deterministischer Prozess betrachtet. In der Praxis enthalten die meisten Builds jedoch eine Reihe nicht deterministischer Elemente: Zeitstempel, zufällige Dateireihenfolgen, unterschiedliche Toolchain-Versionen oder die Auflösung externer Abhängigkeiten. Aufgrund dieser Inkonsistenzen können Nutzende oder Prüfer selbst dann nicht verifizieren, ob das erhaltene Binary tatsächlich aus diesem Quellcode gebaut wurde, wenn Entwickelnde ihren Quellcode veröffentlichen. Das schafft einen blinden Fleck zwischen „dem, was geprüft wurde“ und „dem, was tatsächlich ausgeliefert wurde“. Wie im vorigen Abschnitt gezeigt, wurde diese Lücke bei realen Angriffen ausgenutzt. Malware-Autoren haben den Build-Prozess selbst angegriffen, um Hintertüren einzuschleusen, indem sie Build-Server kompromittierten, Compiler manipulierten oder die Auflösung von Abhängigkeiten veränderten. Reproduzierbare Builds sollen diese Lücke schließen. Indem sie Determinismus beim Build erzwingen, ermöglichen sie unabhängigen Parteien wie Prüfern, Forschenden oder Watchdog-Organisationen, die Software selbst zu kompilieren und zu verifizieren, dass das entstandene Binary mit dem an die Nutzenden verteilten identisch ist. Diese kryptografische Übereinstimmung belegt, dass der Build-Prozess nicht manipuliert wurde und das Binary den geprüften Quellcode widerspiegelt.

Im gesamten Open-Source-Ökosystem haben reproduzierbare Builds bereits erheblich an Bedeutung gewonnen. Projekte wie Debian, Tor und die Bitcoin-Core-Wallet haben eine Infrastruktur für reproduzierbare Builds eingeführt, um sicherzustellen, dass die Nutzenden genau das ausführen, was die Entwickelnden beabsichtigt haben. Diese Praktiken werden inzwischen in Initiativen wie dem Reproducible Builds Project [8] formalisiert und von Normungsgremien und Behörden gefördert.

Während dieses Konzept bei Desktop- und Server-Software erheblich an Bedeutung gewonnen hat, ist die Anwendung dieser Prinzipien auf mobile Apps deutlich komplexer. Mobile Plattformen bringen besondere technische Einschränkungen, fragmentierte Ökosysteme und plattformspezifisches Verhalten mit sich, die es erheblich erschweren, Reproduzierbarkeit zu erreichen.

Herausforderungen beim Reproduzieren mobiler App-Builds

Das Reproduzieren mobiler App-Builds stellt aufgrund der Komplexität und Variabilität mobiler Entwicklungsumgebungen besondere Herausforderungen. Toolchains wie Android Studio oder Xcode entwickeln sich rasch weiter, und schon geringfügige Unterschiede bei Compiler-Versionen oder Build-

Konfigurationen können zu abweichenden Ergebnissen führen. Mobile Builds betten in das endgültige Binary typischerweise nicht deterministische Metadaten ein, etwa Zeitstempel, Versionscodes und Build-Kennungen. Identische Rebuilds sind dadurch ohne Bereinigung unmöglich.

Paketierungswerkzeuge führen durch ZIP-Kompression und Dateireihenfolge weitere Variationen ein. Systeme zur Verwaltung von Abhängigkeiten (z. B. Maven, CocoaPods) können nicht gepinnte oder inkonsistente Bibliotheken beziehen, was zu unbemerkten Code-Änderungen führt. Android-Builds nutzen außerdem Obfuskierungswerkzeuge wie ProGuard und R8, die Binaries abhängig von der Build-Umgebung verändern. Zusammengenommen machen diese Faktoren die Reproduzierbarkeit bei mobilen Apps zu einem technisch anspruchsvollen Ziel.

Trotz dieser Herausforderungen hat das Android-Ökosystem bemerkenswerte Fortschritte in Richtung reproduzierbarer Builds gemacht. Die Plattform F-Droid erzwingt für Open-Source-Apps eine Rebuild-Verifikation, unterstützt durch Werkzeuge wie Reproducible Builds for Android (RBA), die Gradle-Builds in Docker und striktes Pinning von Abhängigkeiten verwenden. Gradle selbst unterstützt inzwischen Reproduzierbarkeit durch Konfigurationsschalter, die Datei-Zeitstempel und die Reihenfolge in Archiven steuern. Entwickelnde können Builds zusätzlich stabilisieren, indem sie Locales, Zeitzonen und Build-Metadaten festlegen. Reproduzierbarkeit ist unter iOS im Vergleich zu Android mit größeren Herausforderungen verbunden. Das liegt vor allem daran, dass die Plattform auf Apples proprietäre Xcode-Toolchain und einen streng kontrollierten Signaturprozess angewiesen ist. Weitere Faktoren wie die Bitcode-Kompilierung und die erneute Signierung durch den App Store können Variabilität einführen, die es erschwert zu verifizieren, dass ein verteiltes Binary exakt seinem ursprünglichen Quellcode entspricht.

2.3. Binary Transparency

Binary Transparency (BT) stärkt die Sicherheit der Software-Lieferkette, indem sie Software-Releases öffentlich prüfbar macht [9]. Im Kern beruht BT auf einem kryptografischen Log, das nur um Einträge ergänzt werden kann und häufig auf einer Merkle-Baum-Struktur aufbaut. Darin wird jedes veröffentlichte Software-Binary zusammen mit Metadaten wie Versionsnummern, Zeitstempeln und Informationen zur Identität der Entwickelnden aufgezeichnet. Diese Logs sind manipulationssicher und öffentlich zugänglich. Sie erlauben es Dritten, unabhängig zu verifizieren, dass ein bestimmtes Binary dem offiziell veröffentlichten und geprüften entspricht.

Wenn Entwickelnde ein neues Binary erzeugen und signieren, reichen sie zusätzlich einen kryptografischen Hash des Binarys und die zugehörigen Metadaten bei einem Transparency Log ein. Das Log stellt einen Signed Entry Timestamp (SET) aus. Dieser belegt, dass die Einreichung stattgefunden hat, und legt die Position des Binarys in der unveränderlichen Log-Historie fest. Dieser Eintrag kann zusammen mit dem Binary selbst verteilt werden, sodass Nutzende, Prüfer oder clientseitige Verifizierer die Aufnahme und Authentizität des Binarys bestätigen können. Unabhängige Monitore prüfen das Log regelmäßig auf Auffälligkeiten, etwa nicht autorisierte Einträge, Inkonsistenzen in der Release-Reihenfolge oder Versuche, eine Version zu unterdrücken oder zurückzusetzen. Prüfer können Binaries außerdem aus dem Quellcode neu bauen und mit den im System protokollierten vergleichen. So ist zusätzlich sichergestellt, dass das, was gebaut wurde, dem entspricht, was signiert und veröffentlicht wurde.

Wichtig ist: BT beruht nicht auf dem Vertrauen in eine einzelne Instanz. Da Logs öffentlich überprüfbar sind und jede Person einen Monitor betreiben oder Audits durchführen kann, dezentralisiert BT die Aufsicht. Sie wandelt die Softwareverteilung von einem System, das auf blindem Vertrauen in Signaturschlüssel, Build-

Server oder App-Stores beruht, in eines, das auf kryptografischem Nachweis und Transparenz gründet.

Indem BT herkömmliche Signaturpraktiken um öffentlich überprüfbare Logs erweitert, führt sie eine neue Verteidigungsebene gegen eine Reihe von Angriffen auf die Software-Lieferkette ein. Sie stellt sicher, dass sich eine Manipulation auch dann nicht verbergen lässt, wenn ein Build-System kompromittiert oder ein Signaturschlüssel missbraucht wird. Jedes veröffentlichte Binary hat eine nachvollziehbare, öffentliche Historie, und jede Unregelmäßigkeit kann erkannt und untersucht werden. BT wirkt außerdem Nötigung oder Zensur in zentralisierten Ökosystemen entgegen. Verteilt ein App-Store unbemerkt ein schädliches oder verändertes Binary, macht das Fehlen eines passenden Log-Eintrags oder eine Abweichung zwischen dem Log und der verteilten Version die Diskrepanz sichtbar.

Sigstore

Sigstore [10] ist ein Open-Source-Projekt, das Werkzeuge und Infrastruktur bereitstellt, um das Signieren und Verifizieren von Software sicherer, transparenter und zugänglicher zu machen. Es erlaubt Entwickelnden, Software-Artefakte wie Binaries, Container-Images und Quellcode zu signieren, ohne langfristige private Schlüssel verwalten zu müssen. Stattdessen verwendet Sigstore kurzlebige Zertifikate, die an vertrauenswürdige Identitätsanbieter (wie GitHub oder Google) gebunden sind, und protokolliert alle Signaturen in einem öffentlichen, manipulationssicheren Transparency Log namens Rekor.

Durch die Kombination aus schlüssellosem Signieren, identitätsbasierter Authentifizierung und öffentlicher Protokollierung trägt Sigstore dazu bei, dass sich Software-Artefakte auf ihren Ursprung zurückführen und von jeder Person verifizieren lassen. Das erschwert es Angreifenden, nicht vertrauenswürdigen oder schädlichen Code in die Lieferkette einzuschleusen. Sigstore ist in der Open-Source-Community weit verbreitet und bildet eine solide Grundlage für die Umsetzung von Binary-Transparency-Systemen.

2.4. Die EUDI Wallet

Die European Digital Identity (EUDI) Wallet ist eine mobile Anwendung, die im Rahmen der EU-Verordnung eIDAS 2.0 entwickelt wird. Sie soll Bürgerinnen und Bürgern eine sichere, standardisierte Möglichkeit bieten, digitale Identitätsnachweise (wie nationale Ausweise, Führerscheine sowie akademische oder Gesundheitsdaten) zu speichern und vorzuzeigen. Die Wallet soll in allen EU-Mitgliedstaaten funktionieren und bringt eine Reihe von Akteuren zusammen (dargestellt in Abbildung 1):

Wallet-Provider sind Behörden oder zertifizierte private Anbieter, die für die Entwicklung und Pflege der Wallet-Anwendungen verantwortlich sind. Sie stellen sicher, dass die Wallet die technischen und regulatorischen Anforderungen von eIDAS 2.0 erfüllt, und betreuen den Software-Lebenszyklus einschließlich Updates und Verteilung. Credential-Aussteller sind Stellen, die überprüfbare Nachweise (Verifiable Credentials) an Wallet-Nutzende ausstellen. Diese Nachweise müssen sicher gespeichert und über die Wallet vorgezeigt werden. Verifizierer sind die Organisationen (Relying Parties oder Service Provider), die über die EUDI Wallet vorgezeigte Nachweise anfordern und validieren. Endnutzende sind die Bürgerinnen und Bürger, die die Wallet verwenden, um ihre digitale Identität zu verwalten und Nachweise online wie offline vorzuzeigen, etwa um ein Bankkonto zu eröffnen, sich an einer Schule einzuschreiben oder grenzüberschreitende öffentliche Dienste zu nutzen. Darüber hinaus sind Behörden der Mitgliedstaaten nationale Regierungen oder benannte Institutionen, die die Ausgabe von Wallets beaufsichtigen, Wallet-Provider zertifizieren und die Einhaltung der EU-Vorschriften durchsetzen.

Die Wallet soll als vertrauenswürdige Schnittstelle dienen, um personenbezogene Identitätsdaten über Grenzen, Institutionen und Anwendungsfälle hinweg zu verwalten. Die Kompromittierung des Build-Prozesses könnte es Angreifenden ermöglichen, Hintertüren in Wallet-Binaries einzufügen, die Diebstahl von Nachweisen, Überwachung oder unbefugten Datenzugriff ermöglichen, ohne dass sich im Quellcode etwas sichtbar ändert.

Angesichts der kritischen Natur dieser Interaktionen und der Tatsache, dass Wallet-Software von mehreren Providern in verschiedenen Rechtsordnungen entwickelt werden kann, reichen herkömmliche Mechanismen wie Code-Signierung und Versionskontrolle allein nicht aus, um Vertrauen, Integrität und Transparenz im gesamten Ökosystem sicherzustellen.

3. Ziele

Dieses Projekt will einen überprüfbaren Verteilungsprozess für die EUDI Wallet schaffen und konzentriert sich dabei auf vier zentrale Ziele:

(i) Prüfbarkeit von Software-Releases: Wallet Binary Transparency (WBT) stellt sicher, dass jedes Wallet-Binary, das von einer nationalen Behörde oder einem zertifizierten Anbieter entwickelt wird, einen überprüfbaren Eintrag hinterlässt, indem jedes Signaturereignis in einem öffentlichen Transparency Log protokolliert wird, das nur um Einträge ergänzt werden kann. Dadurch entsteht ein kryptografischer Prüfpfad, mit dem jede Person, darunter Entwickelnde, Prüfer, Regulierungsbehörden und Endnutzende, die Authentizität eines bestimmten Releases verifizieren und nicht autorisierte oder manipulierte Versionen erkennen kann.

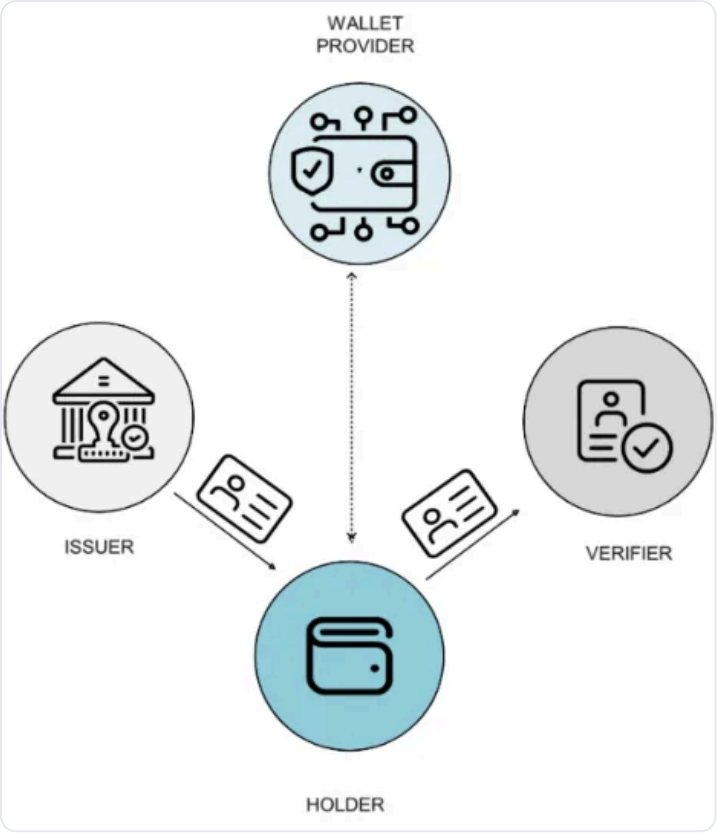


Abbildung 1: Das Wallet-Ökosystem

(ii) Integration in automatisierte Build-Workflows: Bei WBT ist die Protokollierung im Transparency Log direkt in die CI/CD-Pipeline eingebettet. Durch die Automatisierung der Signierung von Binaries und der Einreichung beim Transparency Log soll WBT einen skalierbaren und über alle Releases hinweg einheitlichen Prozess erreichen. Jeder Build wird mit signierten Metadaten und einem eindeutigen, mit Zeitstempel versehenen Log-Eintrag verknüpft. Das gewährleistet Nachvollziehbarkeit über den gesamten Software-Lebenszyklus.

(iii) Schutz der Endnutzenden durch Verifikation bei der Installation: Um Nutzende im Moment der Installation zu schützen, prüft ein Verifikationsmodul, ob das Binary ordnungsgemäß protokolliert wurde. Es ruft Inclusion Proofs aus dem Transparency Log ab und validiert die Integrität und Signatur des Binarys. Werden Inkonsistenzen festgestellt, etwa ein fehlender oder veränderter Log-Eintrag, wird die Installation blockiert oder zur Prüfung markiert.

(iv) Übereinstimmung mit regulatorischen Anforderungen: Durch die Kombination aus öffentlichen Transparency Logs, Attestierungen reproduzierbarer Builds und unabhängiger Verifikation bietet das System eine starke Grundlage, um die Einhaltung der Anforderungen von eIDAS 2.0 nachzuweisen. Es sichert nicht nur die Software-Lieferkette, sondern unterstützt auch die regulatorische Aufsicht durch überprüfbare Nachweise für die Integrität und Authentizität der Software.

Zusammen tragen diese Ziele zu einem Verteilungsmodell bei, das nicht nur von Grund auf sicher, sondern auch unabhängig prüfbar ist und die hohen Sicherheitsstandards erfüllt, die für digitale Identitätssysteme in der EU erforderlich sind.

4. Wallet Binary Transparency

4.1. Akteure im Ökosystem

Dieser Abschnitt behandelt den Entwurf eines Binary-Transparency-Systems für die EUDI Wallet. Zunächst werden die Rollen der verschiedenen Akteure im Wallet-Verteilungsökosystem definiert. Jeder Akteur erfüllt bestimmte Aufgaben, die in unterschiedlichen Phasen des Software-Lebenszyklus zu Überprüfbarkeit und Transparenz beitragen. Das System ist so gestaltet, dass Vertrauen und Verantwortung auf diese unabhängigen Beteiligten verteilt werden.

Wallet-Provider: Sobald eine neue Version zur Veröffentlichung bereit ist, kompilieren sie den Quellcode zu einem Binary und signieren es digital mit kryptografischen Schlüsseln. Wichtig ist, dass Entwickelnde außerdem Metadaten zum Binary, darunter Hash, Version und Signatur, beim Binary Transparency Log einreichen müssen. Zusätzlich können Entwickelnde Anleitungen für reproduzierbare Builds veröffentlichen. Damit können andere unabhängig verifizieren, dass das verteilte Binary dem vorgesehenen Quellcode entspricht. Außerdem überwachen Entwickelnde das Transparency Log, um nicht autorisierte Einträge zu erkennen, die auf Schlüsselmissbrauch oder Identitätsvortäuschung hindeuten können (oder übertragen diese Aufgabe alternativ einem Monitor).

App-Vertriebsplattformen (App-Stores) sind die primären Kanäle, über die Wallet-Anwendungen an Endnutzende ausgeliefert werden. Diese Stores machen die Wallet-Software zugänglich und setzen dabei plattformspezifische Richtlinien zu Signierung, Paketierung und Update-Auslieferung durch. In unserem System bleiben App-Stores der Auslieferungskanal, doch die Integrität dessen, was sie verteilen, kann nun unabhängig geprüft werden. Monitore oder Clients können das Binary des App-Stores mit dem öffentlichen

Log vergleichen und seine Authentizität verifizieren. Der Log-Betreiber unterhält die Kerninfrastruktur des Systems: das kryptografisch überprüfbare Log, das nur um Einträge ergänzt werden kann. Dieses Log zeichnet jedes eingereichte Wallet-Binary auf und stellt einen Signed Entry Timestamp (SET) aus, um die Aufnahme zu bestätigen. Das Log wird mithilfe von Merkle-Bäumen implementiert, wodurch alle Einträge unveränderlich und manipulationssicher sind. Der Betreiber muss Verfügbarkeit, Konsistenz und korrekten Betrieb des Logs gewährleisten und auf Anfrage Inclusion Proofs und Consistency Proofs an Clients und Monitore ausliefern. Die Handlungen der Log-Betreiber werden fortlaufend von Dritten geprüft.

Unabhängige Rebuilder beziehen den Quellcode und die Build-Konfiguration der Wallet aus öffentlichen Repositories und versuchen, das verteilte Binary zu reproduzieren. Stimmt der resultierende Hash mit dem im Transparency Log aufgezeichneten überein, können sie eine „Reproduzierbarkeitsbestätigung" veröffentlichen. Diese Bestätigungen sind starke Signale dafür, dass der Build-Prozess ehrlich und deterministisch war. Werden Abweichungen festgestellt, lösen Rebuilder Warnungen aus, die auf mögliche Manipulation oder fehlende Reproduzierbarkeit hinweisen.

Endnutzende, oder Holder, interagieren mit dem System in der letzten Stufe der Verteilungskette. Wird eine neue Wallet-Version installiert, prüft ein Client-Verifikationsmodul die Signatur, ruft den entsprechenden Eintrag aus dem Transparency Log ab und validiert den Inclusion Proof. In fortgeschritteneren Konfigurationen kann es auch prüfen, ob das Release von Rebuildern unabhängig bestätigt wurde. Schlagen die Prüfungen fehl, kann das Gerät die Nutzenden warnen oder die Installation blockieren. Dieser Entwurf befähigt Endnutzende, die Integrität der Software zu verifizieren.

Monitore sind unabhängige Stellen, die für die Überprüfung der Integrität einzelner Einträge im Transparency Log verantwortlich sind. Ihre Hauptaufgabe ist es, Anzeichen von Fehlverhalten wie fehlende, veränderte oder nicht autorisierte Einträge zu erkennen, indem sie bestimmte Einreichungen beim Log untersuchen. Sie rufen regelmäßig Einträge ab, validieren deren Inclusion Proofs und Signed Entry Timestamps (SETs) und stellen sicher, dass jedes protokollierte Artefakt dem entspricht, was verteilt wurde oder erwartet wird. Werden Unregelmäßigkeiten festgestellt, etwa ein signiertes Binary, das nicht ordnungsgemäß protokolliert wurde, oder eine Abweichung zwischen einem Log-Eintrag und einem veröffentlichten Binary, können Monitore Warnungen auslösen, Entwickelnde benachrichtigen oder regulatorische Maßnahmen anstoßen.

Prüfer (Auditoren) überwachen den Log-Anbieter, um sicherzustellen, dass er Informationen ordnungsgemäß anfügt und eine konsistente globale Sicht aufrechterhält. Prüfer spielen eine Schlüsselrolle bei der Erkennung unehrlichen Verhaltens des Log-Anbieters. Jeder Prüfer kann erkennen und unwiderlegbar nachweisen, ob ein Log-Anbieter betrogen hat. Da jede Person aus der Öffentlichkeit ohne besonderen Zugang als Prüfer auftreten kann, müssen Log-Anbieter unter der Annahme arbeiten, dass ihre veröffentlichten Daten fortlaufend überwacht und geprüft werden. Diese Konstellation erzwingt zusammen mit einem sicheren Konsistenzprotokoll zum Austausch von Log-Fingerprints ehrliches Verhalten des Log-Anbieters.

Regulierungsbehörden wie die Europäische Kommission oder nationale Cybersicherheitsbehörden überwachen die Einhaltung des eIDAS-2.0-Rahmenwerks. Sie können die Einhaltung von Binary-Transparency-Praktiken als Teil von Zertifizierungs- und Sicherungsprozessen für Software vorschreiben. Behörden können auch Monitore betreiben, Reproduzierbarkeit verifizieren, Meldepflichten durchsetzen

oder Vorfälle im Zusammenhang mit kompromittierter oder nicht konformer Wallet-Software untersuchen. Ihre Beteiligung stellt sicher, dass rechtliche und verfahrensbezogene Schutzmaßnahmen die technischen Garantien des Systems ergänzen.

Zusammen bilden diese Akteure ein dezentrales, aber vernetztes Ökosystem, das eine sichere, transparente und prüfbare Verteilung von Wallet-Software unterstützt, wie in Abbildung 2 dargestellt. Jeder Akteur trägt unabhängig zu einer gemeinsamen Sicherheitslage bei. So ist gewährleistet, dass kein einzelner Vertrauenspunkt das System untergraben kann und die Integrität der digitalen Identität in der EU für alle überprüfbar bleibt.

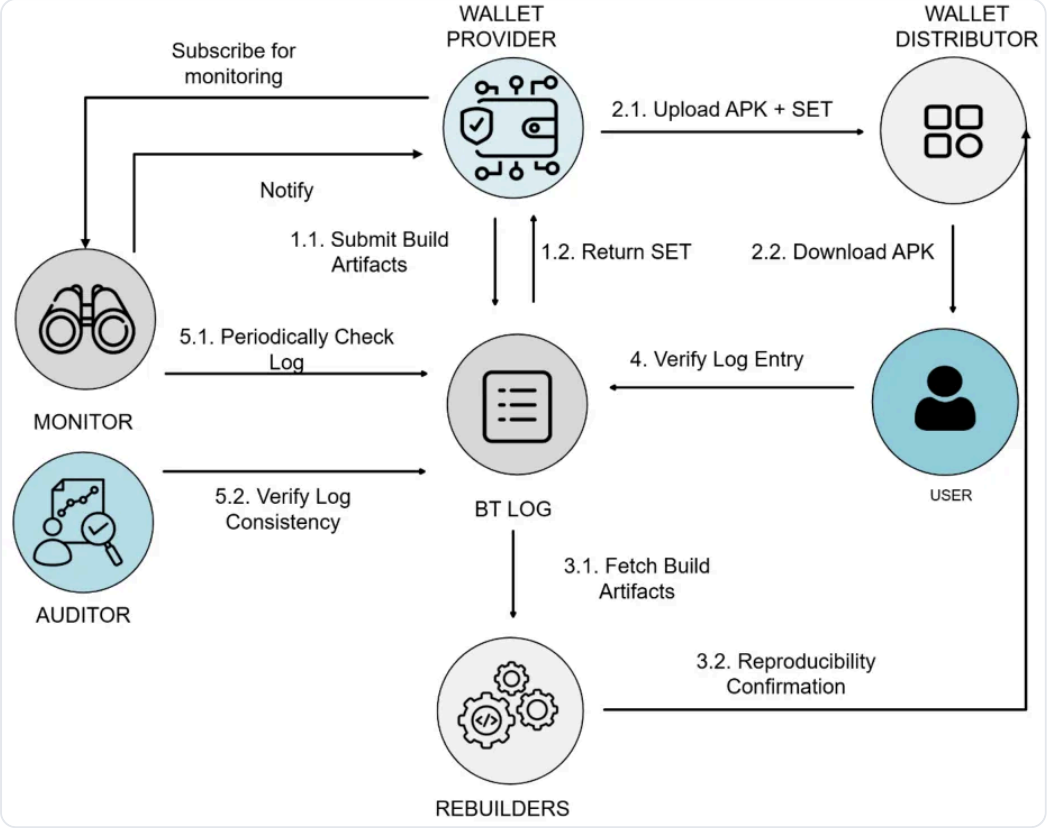


Abbildung 2: Übersicht über Wallet Binary Transparency

4.2. Entwurf von Wallet Binary Transparency

Das Wallet-Binary-Transparency-System, inspiriert von Mobile App Distribution Transparency (MADT) [11], ist in eine strukturierte Reihe ineinandergreifender Phasen gegliedert:

Phase 1: Einreichung des Releases durch den Wallet-Provider

In dieser ersten Phase bereiten Wallet-Entwickelnde ein neues Release vor, indem sie die Wallet-Anwendung aus dem Quellcode kompilieren. Vor der Verteilung signieren die Entwickelnden das entstandene Binary mit einem genehmigten kryptografischen Schlüssel, sodass sich der Ursprung des Builds authentifizieren lässt. Entscheidend ist, dass das signierte Binary nicht sofort verteilt wird. Stattdessen reichen die Entwickelnden eine Build Attestation bei einem öffentlich zugänglichen Binary Transparency Log ein. Diese Attestation enthält den Binary-Hash, Versionsmetadaten, die Signatur und eine eindeutige Kennung.

Nach der Einreichung fügt das Transparency Log den Eintrag seiner Merkle-Baum-Datenstruktur an und stellt einen Signed Entry Timestamp (SET) aus. Dieser SET dient als manipulationssicherer Beleg dafür, dass das Release im Log aufgezeichnet wurde. Protokoll 1 fasst diese Phase zusammen.

1. Die Wallet-Entwicklung (WD) kompiliert die Wallet-Anwendung aus dem Quellcode in einer reproduzierbaren Build-Umgebung und erzeugt ein Binary-Artefakt (z. B. wallet.apk).
2. WD signiert das Binary mit einem genehmigten kryptografischen Schlüssel (oder einem Sigstore-Zertifikat) und erzeugt dabei eine losgelöste Signatur (detached signature).
3. WD erzeugt eine Build Attestation, die einen kryptografischen Hash des Binarys, Versionsmetadaten (z. B. Release-Version, Build-Zeitstempel), die Signatur und die Schlüsselidentität enthält.
4. WD reicht die Build Attestation per authentifizierter Anfrage beim Log-Betreiber ein.
5. WBT fügt die Attestation als neues Blatt (leaf) in seinen Merkle-Baum ein und gibt einen Signed Entry Timestamp (SET) an WD zurück. Der SET enthält den Zeitpunkt der Aufnahme, den Root-Hash des Merkle-Baums und eine kryptografische Signatur des Log-Betreibers.
6. WD paketiert das Binary, seine Signatur und den SET für das Release. Das Binary wird Endnutzenden erst zur Verfügung gestellt, wenn der SET empfangen und in die Release-Metadaten aufgenommen wurde.

Protokoll 1: Protokoll zur Einreichung eines Releases durch die Entwickelnden Dieses Protokoll stellt sicher, dass jedes Release kryptografisch signiert, öffentlich aufgezeichnet und vor der Verteilung überprüfbar ist.

Phase 2: Verteilung und Veröffentlichung der Wallet

Nach erfolgreicher Einreichung beim Log werden das signierte Binary und der zugehörige SET über offizielle Verteilungskanäle veröffentlicht, etwa nationale App-Stores, Regierungsportale oder zugelassene Plattformen Dritter. Da der Build bereits protokolliert wurde, dient diese Phase ausschließlich der Verbreitung, nicht der Verifikation.

Zur Unterstützung der Transparenz können Entwickelnde zusätzlich Quellcode, Build-Skripte, Abhängigkeitsmanifeste und Umgebungsspezifikationen veröffentlichen, um Reproduzierbarkeit zu ermöglichen. Die Verteilungsmetadaten sollten einen Verweis auf den Log-Eintrag enthalten (z. B. Log-Index oder Hash), damit nachgelagerte Verifizierer den entsprechenden Eintrag leicht finden können.

1. Die Wallet-Entwicklung (WD) veröffentlicht das Release-Bundle über zugelassene App-Vertriebsplattformen.

2. Zur Unterstützung von Transparenz und reproduzierbaren Builds kann WD zusätzlich veröffentlichen:

    - Quellcode (oder einen Referenz-Commit in einer öffentlichen Versionskontrolle)
    - Build-Skripte (z. B. Gradle, Dockerfile)
    - Abhängigkeits-Lock-Dateien (z. B. dependencies.lock)
    - Umgebungsspezifikationen (z. B. Toolchain-Versionen, Betriebssystemkonfigurationen)

3. WD nimmt in die veröffentlichten Metadaten einen eindeutigen Verweis auf den Eintrag im Transparency Log auf. So können Endnutzende, Prüfer oder automatisierte Systeme den entsprechenden Log-Datensatz aus dem Wallet Binary Transparency Log abrufen und verifizieren.

Protokoll 2: Protokoll zur Verteilung und Veröffentlichung der Wallet

Phase 3: Unabhängiges Neubauen und Bestätigung

In dieser Phase laden unabhängige Rebuilder den Quellcode herunter und versuchen, das verteilte Binary mit der veröffentlichten Build-Umgebung zu reproduzieren. Dieser Schritt verifiziert, dass das Binary aus dem erwarteten Quellcode ohne versteckte Änderungen kompiliert wurde.

Entspricht das neu gebaute Binary dem Original (Hash-Gleichheit), veröffentlicht der Rebuilder eine Reproduzierbarkeitsbestätigung (Reproducibility Confirmation). Diese Bestätigung enthält den Build-Hash, den Verweis auf den ursprünglichen Log-Eintrag und eine kryptografische Signatur des Rebuilders. Mehrere unabhängige Bestätigungen sind ein starker Beleg dafür, dass das Binary korrekt gebaut und während der Kompilierung nicht manipuliert wurde.

Liefert der Rebuild keinen übereinstimmenden Hash, kann der Rebuilder stattdessen einen Mismatched Build Alert einreichen. Dieser wird ebenfalls protokolliert und kann weitere Untersuchungen auslösen oder die Auslieferung durch nachgelagerte Systeme stoppen.

1. Der unabhängige Rebuilder (IR) beschafft die notwendigen Materialien:
    - Quellcode am angegebenen Commit
    - Build-Skripte, Abhängigkeitsmanifeste und Toolchain-Konfigurationen
    - Verweis auf den ursprünglichen Log-Eintrag (UUID oder Log-Index)
    - Offizielles Binary (für den Hash-Vergleich)
2. IR baut das Binary mit der bereitgestellten Umgebung in einem sauberen, isolierten Kontext (z. B. Docker, Nix). Das Ergebnis ist ein Kandidaten-Binary zur Verifikation.
3. IR berechnet den Hash des neu gebauten Binarys und vergleicht ihn mit dem im ursprünglichen Rekor-Eintrag aufgezeichneten Hash.
    - Stimmen die Hashes überein → der Build ist reproduzierbar
    - Weichen die Hashes ab → eine Abweichung wird markiert

4.a. IR erzeugt und signiert eine Reproduzierbarkeitsbestätigung, die Folgendes enthält:

- Hash des neu gebauten Binarys
- Verweis auf den ursprünglichen Log-Eintrag
- Zeitstempel und Identität des Rebuilders
- Eine digitale Signatur, die diese Elemente bindet

4.b. Stimmt das neu gebaute Binary nicht überein, reicht IR einen Mismatched Build Alert bei Monitoren oder Regulierungsstellen ein.

Protokoll 3: Protokoll für unabhängiges Neubauen und Bestätigung

Phase 4: Client-Verifikation bei der Installation

Wenn Nutzende die EUDI Wallet auf ihrem Gerät installieren oder aktualisieren, wird das eingebettete Verifikationsmodul aktiviert. Diese Komponente führt vor der Installation eine Reihe automatisierter Prüfungen durch:

Nur wenn alle Prüfungen bestanden sind, fährt das System mit der Installation fort. Schlägt ein Verifikationsschritt fehl, etwa wegen eines fehlenden Log-Eintrags, eines ungültigen SET oder fehlender unabhängiger Bestätigungen, werden die Nutzenden gewarnt. Die Installation wird je nach Richtlinieneinstellung angehalten oder ganz blockiert.

1. Bei Installation oder Update der EUDI Wallet wird das eingebettete Verifikationsmodul (VM) auf dem Nutzergerät (UD) automatisch aktiviert.
2. VM prüft die digitale Signatur des Binarys anhand einer vertrauenswürdigen Liste öffentlicher Entwicklerschlüssel oder autorisierter Signaturidentitäten. Ist die Signatur ungültig oder der Signierende nicht vertrauenswürdig, schlägt der Vorgang sofort fehl.
3. VM ruft den zum Binary gehörenden Signed Entry Timestamp (SET) oder die UUID ab und fragt das Wallet Binary Transparency Log ab
    - Es ruft den Merkle Inclusion Proof für den angegebenen SET ab
    - Es verifiziert, dass Binary-Hash und Metadaten mit dem aufgezeichneten Log-Eintrag übereinstimmen
    - Es bestätigt die Signatur des SET mit dem öffentlichen Schlüssel von WBT.
4. Je nach Richtlinie prüft VM optional im Log, ob Reproduzierbarkeitsbestätigungen zur selben Log-UUID vorliegen.

5.a. Bestehen alle Prüfungen, wird die Installation fortgesetzt.

5.b. Schlägt eine Prüfung fehl (z. B. fehlender SET, ungültige Signatur oder unzureichende Bestätigungen), je nach Client-Richtlinie:

- Das Verifikationsmodul hält die Installation an oder blockiert sie
- Den Nutzenden wird eine Warnung oder ein Alarm angezeigt

Protokoll 4: Protokoll zur Client-Verifikation bei der Installation

Phase 5: Kontinuierliche Überwachung und Prüfung

Die letzte Phase läuft fortlaufend und gilt über den gesamten Lebenszyklus aller Wallet-Releases. Monitore beobachten das Transparency Log kontinuierlich, um sicherzustellen, dass es sich korrekt verhält: Es fügt neue Einträge in der richtigen Reihenfolge an, liefert gültige Consistency Proofs und erhält für alle Clients eine einheitliche Sicht auf das Log.

Erkennt ein Monitor Unregelmäßigkeiten, kann er automatisierte Warnungen auslösen, Regulierungsbehörden benachrichtigen, öffentliche Warnungen veröffentlichen oder Verfahren für Software-Rollback und Widerruf einleiten. Monitore dienen somit als Frühwarnmechanismus des Systems und stärken die kollektive Wachsamkeit.

1. Monitore fragen WBT kontinuierlich in regelmäßigen Abständen ab, um neu angefügte Einträge abzurufen.
2. Monitore vergleichen beobachtete Log-Einträge mit den erwarteten Release-Daten und Richtlinien-Baselines:
    - Bestätigt, dass jedes Release von autorisierten Entwickelnden signiert wurde
    - Markiert unerwartete Signaturidentitäten oder die Wiederverwendung von Schlüsseln über nicht zusammenhängende Releases hinweg
    - Log-Antworten stimmen mit dem bekannten, vertrauenswürdigen öffentlichen Schlüssel von WBT überein
3. Prüfer fragen WBT kontinuierlich ab, um aktualisierte Log Checkpoints und Consistency Proofs zwischen Checkpoints abzurufen. Prüfer verifizieren, dass:
    - Jeder Checkpoint gültig und mit früheren Zuständen konsistent ist
    - Verschiedenen Clients keine gespaltenen Sichten oder Equivocation (d. h. widersprüchliche Checkpoints) präsentiert werden
4. Bei Bedarf stoßen Monitore und Prüfer Reaktionsmaßnahmen der Regulierungsbehörden an.

Protokoll 5: Protokoll für kontinuierliche Überwachung und Prüfung

5. Überlegungen zur Umsetzung

Der folgende Abschnitt skizziert die fünf Systemphasen und beschreibt, wie sie sich konkret mit der EUDI Wallet Reference Implementation für Android und der Open-Source-Infrastruktur von Sigstore, einschließlich Cosign und Rekor, umsetzen lassen.

Phase 1: Signierung durch die Entwickelnden und Einreichung beim Log

Die erste Phase des Systems beginnt in der CI/CD-Build-Pipeline. Sobald Wallet-Entwickelnde eine neue Version der Anwendung erzeugen, in unserer Umsetzung ein Android-APK, müssen sie das Binary vor der Veröffentlichung signieren. Dieser Schritt kann entweder mit einem langlebigen, manuell verwalteten

privaten Schlüssel oder mit dem Werkzeug Cosign von Sigstore für schlüsselloses Signieren erfolgen. Im zweiten Modell authentifiziert Cosign die Entwickelnden über einen OpenID-Connect-Anbieter (OIDC-Anbieter, z. B. GitHub oder Google) und erhält von Fulcio ein kurzlebiges Zertifikat, mit dem das Binary signiert wird.

Sobald das Binary signiert ist, reicht Cosign automatisch ein signiertes Metadaten-Bundle bei Rekor, dem Transparency Log von Sigstore, ein. Dieses Bundle enthält den Digest des Binarys, die Signatur, das Identitätszertifikat der Entwickelnden und weitere Metadaten. Rekor weist dem Log-Eintrag einen Signed Entry Timestamp (SET) zu, im Wesentlichen einen kryptografischen Beleg, der bestätigt, dass das Binary unveränderlich in einem Merkle-Baum aufgezeichnet wurde, der nur um Einträge ergänzt werden kann. Der SET oder die eindeutige Kennung des Eintrags (UUID) wird aufbewahrt und zusammen mit dem Binary verteilt.

Dieser Prozess lässt sich in der CI-Pipeline der EUDI Wallet Reference Implementation vollständig automatisieren. In einem GitHub-Actions-Workflow können Entwickelnde beispielsweise Cosign-Befehle einbinden, um das gebaute Wallet-Binary zu signieren und die entstandene Signatur bei Rekor einzureichen. Der SET wird in ein Release-Manifest eingebettet, das das veröffentlichte Release begleitet.

Phase 2: Verteilung und Veröffentlichung

Das signierte Binary, die losgelöste Signatur und die zugehörigen transparenzbezogenen Metadaten (einschließlich der Rekor-UUID oder des SET) werden über GitHub Releases, nationale eIDAS-Portale oder vertrauenswürdige, in der EU gehostete Endpunkte veröffentlicht. Alle Artefakte müssen auf den zugehörigen Rekor-Log-Eintrag verweisen. Zur Unterstützung der Verifikation der Reproduzierbarkeit werden außerdem Quellcode, Build-Anleitungen und Manifeste mit gepinnten Abhängigkeiten veröffentlicht. Diese Materialien liegen im selben Repository oder sind aus den Release-Metadaten verlinkt.

Phase 3: Unabhängiges Neubauen und Bestätigung

Prüfer und Rebuilder klonen den Quellcode am angegebenen Commit, stellen die Umgebung mit den bereitgestellten Containerisierungswerkzeugen wieder her und bauen das Binary neu. Stimmt der resultierende Hash mit dem in Rekor protokollierten überein (mithilfe von Werkzeugen wie diffoscope), signiert der Rebuilder eine Reproduzierbarkeitsbestätigung, die auf den ursprünglichen Log-Eintrag verweist.

Phase 4: Client-Verifikation bei der Installation

Das Client-Verifikationsmodul:

- Validiert die Cosign-Signatur anhand einer vertrauenswürdigen Liste öffentlicher Entwicklerschlüssel oder Identitäten (die als Vertrauensanker für das Ökosystem enthalten sind).
- Fragt die Rekor-Instanz ab, um die Aufnahme des Binarys (per UUID oder SET) zu verifizieren und den Inclusion Proof zu erhalten.
- Prüft optional, ob Reproduzierbarkeitsbestätigungen vertrauenswürdiger Rebuilder vorliegen.

Die Verifikationslogik ist in die mobile Codebasis eingebettet. Fehlgeschlagene Verifikationen (fehlender Log-Eintrag, ungültige Signatur oder fehlende Rebuild-Bestätigungen) lösen Durchsetzungsmaßnahmen aus, etwa das Blockieren der Installation oder eine Warnung. Unter Android kann die Verifikation nach dem Download und vor der Installation ausgelöst werden.

Phase 5: Kontinuierliche Überwachung

Unabhängige Monitore fragen das Rekor-Log regelmäßig über rekor-monitor ab, um neue Einträge abzurufen, nachzuverfolgen, welche Identitäten Releases signieren, und unerwartete Wiederverwendung von Schlüsseln zu erkennen.

Reproduzierbarkeit des Builds

Um die Reproduzierbarkeit des Builds der EUDI Wallet Reference Implementation für Android zu bewerten, verwenden wir eine strukturierte und automatisierte Methodik, die Konsistenz über Builds hinweg sicherstellt und jegliche Abweichungen sichtbar macht. Der Prozess gliedert sich in vier zentrale Stufen:

Wir beginnen mit dem Aufbau einer kontrollierten Build-Umgebung mit Docker. Damit lassen sich alle Build-Abhängigkeiten, einschließlich Android SDK, Gradle und Java-Toolchain, in einem Container kapseln und so von Unterschieden des Host-Systems isolieren. Um die Variabilität weiter zu verringern, sind alle Abhängigkeiten explizit auf Versionen festgelegt (versionsgepinnt). Unterschiede durch wechselnde Bibliotheks- oder Plugin-Versionen entfallen damit.

Als Nächstes normalisieren wir die Build-Eingaben, um Quellen von Nicht-Determinismus zu entfernen. Wir konfigurieren die Umgebung mit SOURCE_DATE_EPOCH und setzen damit einen festen Build-Zeitstempel, der zeitbasierte Variabilität neutralisiert. Außerdem passen wir Build-Werkzeuge so an, dass keine flüchtigen Daten wie absolute Dateipfade oder systemspezifische Metadaten einfließen, die andernfalls die Ausgabe des Binarys beeinflussen können.

Der Build-Prozess selbst ist automatisiert und wiederholbar. Er wird durch ein Docker-basiertes Skript gesteuert, das die Android-Wallet in einem sauberen Zustand aus dem Quellcode kompiliert. Diese Automatisierung stellt sicher, dass jeder Aufruf des Build-Prozesses exakt dieselbe Abfolge von Schritten und dieselbe Konfiguration verwendet.

Schließlich führen wir iterative Tests und Vergleiche durch. Der Build wird mehrfach unter denselben kontrollierten Bedingungen ausgeführt, und die resultierenden APK-Dateien werden mit diffoscope verglichen, einem spezialisierten Werkzeug zur Analyse von Binärunterschieden. Dieser Schritt lässt selbst subtile Abweichungen zwischen Builds erkennen und validiert, ob die Ausgabe Bit für Bit identisch und damit tatsächlich reproduzierbar ist.

Rekor-Log-Einträge

Rekor-Log-Einträge enthalten strukturierte Metadaten, die zentrale Informationen über ein signiertes Software-Artefakt oder eine Attestation erfassen. Die Felder eines Rekor-Log-Eintrags werden durch seinen Eintragstyp definiert (z. B. hashedrekord, intoto, rekord usw.), alle Einträge teilen jedoch eine gemeinsame Struktur mit kryptografischen Nachweisen, Zeitstempeln und Signaturmaterial. Für die Prüfungen der Build-Reproduzierbarkeit haben wir uns auf Build Attestations konzentriert.

Build Attestations in Rekor sind strukturierte Metadateneinträge, die beschreiben, wie ein Software-Artefakt erzeugt wurde. Sie enthalten kryptografische Belege des Build-Prozesses und werden üblicherweise zusammen mit einem signierten Artefakt eingereicht, um eine überprüfbare Verbindung zwischen Quellcode, Build-Umgebung und resultierendem Binary herzustellen. Tabelle 1 gibt einen Überblick über die wichtigsten Felder einer Build Attestation in Rekor:

Das Verifikationsmodul auf dem Client-Gerät ruft später die Build Attestation ab, prüft den Hash des Binarys, validiert die Signaturen und bestätigt, dass der Build-Prozess mit dem übereinstimmt, was in der Aussage (statement) angegeben ist.

Skalierbarkeitsüberlegungen zum Transparency Log

Um eine EU-weite Transparenz der Verteilung der EUDI Wallet zu unterstützen, muss ein auf Rekor von Sigstore basierendes WBT-Log skalieren können, um Verifikationsanfragen von bis zu 400 Millionen Nutzenden zu bewältigen. Das Log wird bei der App-Installation, bei Updates oder bei automatisierten Prüfungen abgefragt, was erhebliche Abfragevolumina und mögliche Lastspitzen an Release-Tagen zur Folge hat. Eine naive Einzelinstanz-Bereitstellung von Rekor wäre in diesem Maßstab unzureichend und würde Leistungsengpässe und Verfügbarkeitsrisiken mit sich bringen.

Um dies zu adressieren, unterstützt Rekor Sharding [12], sodass Logs in mehrere unabhängige Merkle-Bäume aufgeteilt werden können. Nach dem Sharding leitet Rekor Einträge und Abfragen automatisch an den richtigen Baum weiter.

Das ermöglicht horizontale Skalierbarkeit, indem die Last auf mehrere durch Merkle-Bäume gestützte Logs verteilt wird, die jeweils unabhängig nur um Einträge ergänzt werden können und abfragbar sind. In der Praxis könnte jeder Shard einem bestimmten Mitgliedstaat oder Wallet-Provider entsprechen, was Konkurrenz um Ressourcen verringert und die Antwortzeiten verbessert. Rekor unterstützt außerdem replizierte Bereitstellungen, bei denen mehrere Serverinstanzen dasselbe Daten-Backend nutzen, um Hochverfügbarkeit und Lastverteilung über geografische Regionen hinweg zu unterstützen.

Für Lesezugriffe mit hohem Durchsatz kann das Log durch statische Tile-APIs [13] gestützt werden. Dies ist eine Technik, die kürzlich bei Certificate-Transparency-Logs eingesetzt wurde, um effizientes CDN-Caching zu ermöglichen und die Last auf den Live- API-Endpunkten von Rekor zu verringern. Werkzeuge wie Trillian-Tessera unterstützen die Veröffentlichung von Merkle-Bäumen auf Tile-Basis. Clients können dadurch Merkle Proofs und Checkpoints über zwischengespeicherte, unveränderliche Blobs abrufen, die auf öffentlichen CDNs liegen, was Latenz und Infrastrukturlast erheblich senkt.

Field	Description
Artifact Digest	The SHA-256 (or similar) hash of the resulting binary, linking the attestation to a specific release.
Build Type	The format or purpose of the attestation, e.g., SLSA provenance v0.2 or in-toto statement.
Builder Identity	The tool or system that performed the build, such as GitHub Actions, a Docker container, or a reproducible CI pipeline.
Source URI and Commit	The Git repository URL and commit hash used as the source for the build.
Build Parameters	Environment variables, tool versions, and other inputs that influence the reproducibility of the build.
Timestamp	The date and time when the build was executed.
Signer Identity	The entity signing the attestation, typically via Cosign, GPG, or another verified identity provider.

Tabelle 1: Felder einer Build Attestation in Rekor

6. Schlussfolgerungen:

Dieses Projekt untersuchte, wie sich Binary Transparency in den Entwicklungs- und Verteilungszyklus der European Digital Identity Wallet (EUDI Wallet) integrieren lässt, um Vertrauen und Verantwortlichkeit zu stärken. Indem das System herkömmliche Signaturpraktiken um unabhängig überprüfbare Logs erweitert und Rebuilds durch Dritte ermöglicht, schützt es vor einer breiten Palette von Bedrohungen der Software-Lieferkette, darunter kompromittierte Build-Pipelines, Missbrauch von Signaturschlüsseln und gezielte Verteilungsangriffe.

Wallet Binary Transparency führt überprüfbare Prüfpfade, unabhängige Rebuild-Bestätigungen, clientseitige Verifikation und kontinuierliche Überwachung ein. Dieses System verlagert das Vertrauen von undurchsichtigen Prozessen und zentralen Stellen auf offene, überprüfbare Mechanismen. Letztlich legt dieser Ansatz das Fundament für eine Infrastruktur für digitale Identitäten, die nicht nur sicher und interoperabel, sondern auch transparent und öffentlich rechenschaftspflichtig ist.

Referenzen

[1] “Operation ShadowHammer | Securelist.” Accessed: May 23, 2025. [Online]. Available: https://securelist.com/operation-shadowhammer/89992/

[2] “CCleanup: A Vast Number of Machines at Risk.” Accessed: May 23, 2025. [Online]. Available: https://blog.talosintelligence.com/avast-distributes-malware/

[3] “The 2025 Software Supply Chain Security Report.” Accessed: May 23, 2025. [Online]. Available: https://www.reversinglabs.com/sscs-report

[4] “ENISA Threat Landscape 2024 | ENISA.” Accessed: May 23, 2025. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

[5] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12223 LNCS, pp. 23–43, 2020, doi:

10.1007/978-3-030-52683-2_2.

[6] “Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies | by Alex Birsan | Medium.” Accessed: May 23, 2025. [Online]. Available: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

[7] K. Thompson, “Reflections on Trusting Trust,” *Commun ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210;GROUPTOPIC:TOPIC:ACM-PUBTYPE>MAGAZINE;WGROUP:STRING:ACM.

[8] “Reproducible Builds — a set of software development practices that create an independently-verifiable path from source to binary code.” Accessed: May 27, 2025. [Online]. Available: https://reproducible-builds.org/

[9] “Binary Transparency.” Accessed: May 23, 2025. [Online]. Available: https://binary.transparency.dev/

[10] “Home · Sigstore.” Accessed: May 27, 2025. [Online]. Available: https://www.sigstore.dev/

[11] M. Lins, R. Mayrhofer, M. Roland, and A. R. Beresford, “Mobile App Distribution Transparency (MADT): Design and Evaluation of a System to Mitigate Necessary Trust in Mobile App Distribution Systems,” pp. 185–203, 2024, doi: 10.1007/978-3-031-47748-5_11.

[12] “Sharding - Sigstore.” Accessed: Jun. 23, 2025. [Online]. Available: https://docs.sigstore.dev/logging/sharding/

[13] “Reflections on a Year of Sunlight - Let’s Encrypt.” Accessed: Jun. 23, 2025. [Online]. Available: https://letsencrypt.org/2025/06/11/reflections-on-a-year-of-sunlight/