



FOM Hochschule für Oekonomie & Management

Hochschulzentrum DLS

Master-Thesis

im Studiengang

Cyber Security Management

zur Erlangung des Grades eines

Master of Science (M.Sc.)

über das Thema

**Design Science Research-geleitete Entwicklung und Evaluation eines
blockchain-basierten Self-Sovereign-Identity-Prototypen mit
Post-Quantum-Kryptografie für kritische Infrastrukturen**

von

Ferris Florian Menzel

Erstgutachter: Prof. Dr. Martin Rupp

Matrikelnummer: 718680

Abgabedatum: 06.01.2026

Vorwort

Die vorliegende Masterarbeit basiert auf dem im Rahmen des „Projektseminar“-Moduls im August 2025 erstellten Exposés. Die dort entwickelte Problemstellung, Forschungsfrage, methodische Konzeption und initiale systematische Literaturrecherche bilden das Fundament dieser Arbeit und wurden für die Masterarbeit vertieft, erweitert und aktualisiert.

Um wissenschaftliche Transparenz, Reproduzierbarkeit und Nachvollziehbarkeit zu gewährleisten dokumentiert der Anhang, entsprechend gekennzeichnet, die vollständige Dokumentation der ersten Iteration der Literaturrecherche des Exposés nach dem PRISMA 2020 Standard, sodass die vorliegende Masterarbeit ein in sich geschlossenes und eigenständiges wissenschaftliches Dokument darstellt.

Abzüglich der Abbildungen, Tabellen und Listings umfasst der Umfang des Hauptteils 75 Seiten.

Gestalterisch folgt die Masterarbeit dem „Leitfaden zur formalen Gestaltung von Seminar- und Abschlussarbeiten (Stand 01/24) erstellt von Prof. Dr. Dr. habil. Clemens Jäger, Prof. Dr. Thomas Kümpel und Prof. Dr. Anja Seng“. Gemäß dieses Leitfadens sieht die vorliegende Masterarbeit keinen Abstract vor. Als Zitations-Standard wird der in der Scientific Community etablierte APA-Zitierstil verwendet.

Der Umfang des Anhangs reflektiert die hohen Anforderungen an wissenschaftliche Gütekriterien hinsichtlich Transparenz und Nachvollziehbarkeit. Hierzu zählen die Dokumentation des Kern-Source-Codes des entwickelten Artefakts sowie die summativen Evaluationsskripte, die eine vollständige Reproduzierbarkeit und Überprüfbarkeit der Ergebnisse ermöglichen. Systematische Querverweise verankern die Elemente des Anhangs im Haupttext und schaffen somit eine integrierte Struktur.

Begleitend zu dieser Masterarbeit liegt eine ZIP-Datei mit allen flüchtigen Quellen und dem vollständigen Source-Code des Artefakts einschließlich der summativen Evaluationsskripte bei.

München, 6.1.2026

Ferris Florian Menzel

Inklusionshinweis

Zur besseren Lesbarkeit wird in dieser Arbeit das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich - sofern nicht anders kenntlich gemacht - auf alle Geschlechter.

Inhaltsverzeichnis

Abbildungsverzeichnis	VIII
Listingverzeichnis	X
Tabellenverzeichnis	XIII
Abkürzungsverzeichnis	XIV
Glossar	XV
1 Einleitung	1
1.1 Problemstellung und Motivation	1
1.2 Stand der Forschung und Identifikation der Forschungslücke	3
1.3 Zielsetzung und Forschungsfragen	5
1.4 Methodisches Vorgehen	6
1.5 Scope und Limitationen	7
1.6 Aufbau der Arbeit	9
2 Theoretische Grundlagen	10
2.1 Self-Sovereign Identity	10
2.1.1 Kernkomponenten der SSI-Architektur	11
2.1.2 Infrastruktur	12
2.1.3 Identifikatoren und kryptographisches Material	12
2.1.4 Credentials und Presentations	13
2.1.5 Wallet-Applikation (Edge Agent und Cloud Agent)	13
2.1.6 Vertrauensmechanismus und Verifikationsprozess	14
2.2 Blockchain-Technologie	14
2.2.1 Klassifizierung von Ledger-Architekturen	15
2.2.2 Die Rolle der Blockchain in SSI-Systemen	16
2.3 Post-Quantum Kryptografie	17
2.3.1 Abgrenzung zu Quantum Cryptography	18
2.3.2 Kategorien von PQC-Algorithmen	18
2.3.3 NIST-Standardisierung der PQC-Algorithmen	19

2.4	Kritische Infrastrukturen und Compliance	20
2.4.1	Regulatorische Anforderungen	20
2.4.2	Compliance-Vorgaben	21
2.5	Kryptoagilität	22
3	Methodik	24
3.1	Systematische Literaturrecherche	24
3.2	Design Science Research	24
3.2.1	Zyklen	26
3.2.2	Artefakte	27
3.2.3	Richtlinien	27
3.3	FEDS-Framework	29
3.3.1	Explikation der Evaluationsziele	29
3.3.2	Wahl der Evaluationsstrategie	30
3.3.3	Bestimmung der zu evaluierenden Eigenschaften	31
3.3.4	Design der individuellen Evaluationsepisoden	31
3.4	DSRM Prozessmodell	33
4	Erste Iteration der Artefaktentwicklung	35
4.1	Designziele dieser Iteration	35
4.2	Anforderungsanalyse	35
4.2.1	Funktionale Anforderungen	35
4.2.2	KRITIS-spezifische Compliance-Anforderungen	36
4.3	Framework- und Technologie-Auswahl	39
4.4	Architekturentwurf	39
4.4.1	Gesamtarchitektur	39
4.4.2	ACA-Py Applikationsarchitektur	41
4.5	Implementierung	43
4.5.1	Zertifikatsstruktur	43
4.5.2	Sidecar-Proxy nginx	44
4.5.3	DLT-Infrastruktur	46
4.5.4	Revocation Registry	47
4.5.5	SSI-Agenten	47
4.5.6	Docker Orchestrierung der Gesamtarchitektur	48
4.6	Formative Evaluation	51
4.6.1	Validierung der Zertifikatkette und ML-DSA-Signaturen	51
4.6.2	Validierung der TLS 1.3 Algorithmen-Aushandlung	52
4.6.3	Validierung der Ledger-Initialisierung	52

4.6.4	Validierung der ACA-Py API-Verfügbarkeit	52
4.6.5	Validierung der Netzwerkisolation	53
4.7	Erkenntnisse und Anpassungsbedarfe	54
4.7.1	Abgleich mit den Designzielen und kritische Erkenntnisse	54
4.7.2	Design-Refinements und Operationalisierung der zweiten Iteration	55
5	Zweite Iteration der Artefaktentwicklung	57
5.1	Designziele dieser Iteration	57
5.2	Architekturentwurf	58
5.2.1	Gesamtarchitektur	58
5.2.2	ACA-Py Applikationsarchitektur	60
5.3	Implementierung	63
5.3.1	Pluginentwicklung: Kryptografie-Abstraktionsschicht	64
5.3.2	Pluginentwicklung: DID-Verarbeitungsschicht	66
5.3.3	Pluginentwicklung: Integration-Patching-Schicht	67
5.3.4	Dockerfile-Modifikation	70
5.3.5	Deployment in docker-compose.yml	72
5.4	Formative Evaluation	72
5.4.1	Validierung des Plugin-Ladevorgangs bei Agent-Start	74
5.4.2	Validierung der Pluginfunktionalität	74
5.5	Finales Artefakt	77
6	Summative Evaluation	79
6.1	Validierung der funktionalen Anforderungen	80
6.1.1	Issuer Discovery	80
6.1.2	Connection Creation	80
6.1.3	Credential Creation	81
6.1.4	Verification with Credentials	82
6.1.5	Credential Revocation	83
6.1.6	Credential Deletion	83
6.2	Validierung der KRITIS-Compliance-Anforderungen	84
6.2.1	Einhaltung spezifischer Parameter-Sets für ML-DSA	84
6.2.2	Einhaltung spezifischer Parameter-Sets für ML-KEM	84
6.2.3	Implementierung hybrider Schlüsseleinigung	85
6.2.4	Bevorzugte Verwendung von TLS 1.3	85
6.2.5	Protokollierung Sicherheitsrelevanter Ereignisse	87
6.2.6	Logische Netzsegmentierung	87
6.2.7	Datenschutz durch Technikgestaltung (Privacy by Design)	87

6.2.8	Grundsatz der Datenminimierung	88
6.2.9	Recht auf Löschung	88
6.3	Validierung der Kryptoagilität	89
6.3.1	Transportlayer	89
6.3.2	Applikationslayer	91
7	Ergebnisse und Diskussion	92
7.1	Beantwortung der Forschungsfragen	92
7.2	Kritische Reflexion	95
7.3	Wissenschaftliche und praktische Beiträge	97
8	Fazit und Ausblick	99
8.1	Zusammenfassung der Ergebnisse	99
8.2	Zukünftige Forschungsrichtungen	100
Anhang		101
Literaturverzeichnis		470
KI-Hilfsmittelverzeichnis		481

Abbildungsverzeichnis

Abbildung 1:	Kontrollverschiebung vom Issuer und Verifier zum User	10
Abbildung 2:	SSI-Systemarchitektur	11
Abbildung 3:	Bestandteile eines DID	12
Abbildung 4:	Blockchain Sequenz	15
Abbildung 5:	Blockchain Klassifizierung	16
Abbildung 6:	Information Systems Research Framework	25
Abbildung 7:	Design Science Research Zyklen	26
Abbildung 8:	Gewählte Evaluationsstrategie im FEDS-Framework	30
Abbildung 9:	DSRM Process Model	33
Abbildung 10:	Gesamtarchitekturentwurf der ersten Iteration	40
Abbildung 11:	ACA-Py High Level Applikationsarchitektur	42
Abbildung 12:	Zertifikatserstellungsworkflow für PQC-basierte Sidecar-Proxies . .	44
Abbildung 13:	Sidecar-Proxy nginx Dockerfile Multi-Stage Build	45
Abbildung 14:	Docker-Compose-Übersicht der ersten Iteration	50
Abbildung 15:	Gesamtarchitekturentwurf der zweiten Iteration	59
Abbildung 16:	ACA-Py High Level Applikationsarchitektur mit PQC-Integration .	62
Abbildung 17:	Aufbau des PQC-Plugins	64
Abbildung 18:	Aufbau des PQC-Plugins - Kryptografie-Abstraktionsschicht	65
Abbildung 19:	Aufbau des PQC-Plugins - DID-Verarbeitungsschicht	66
Abbildung 20:	Aufbau des PQC-Plugins - Integration-Patching-Schicht	68
Abbildung 21:	ACA-Py Multi-Stage Build Dockerfile mit PQC-Integration	71
Abbildung 22:	Docker-Compose-Übersicht der zweiten Iteration	73
Abbildung 23:	TLS 1.3 Enforcement blockiert Legacy-Verbindungsversuch	86
Abbildung 24:	TLS 1.3 Kryptoagiler Fallback von X25519ML-KEM-768 zu X25519	90
Abbildung A-1:	Erste Iteration - Ergebnis der EBSCO Suchanfrage	115
Abbildung A-2:	Erste Iteration - PRISMA 2020 Flussdiagramm	117
Abbildung A-3:	Zweite Iteration - Ergebnis der EBSCO Suchanfrage	171
Abbildung A-4:	Zweite Iteration - PRISMA 2020 Flussdiagramm	173
Abbildung A-5:	Pod mit Sidecar-Proxy	206
Abbildung A-6:	Setup der Entwicklungsumgebung	207
Abbildung A-7:	Cipher Mismatch der TLS-1.3-Verbindung des Webservers	258
Abbildung A-8:	Erfolgreiche Validierung des ML-DSA-Zertifikats des Issuers	261
Abbildung A-9:	Erfolgreiche Validierung der TLS-1.3-Verbindung des Issuers	263
Abbildung A-10:	Erfolgreiche Validierung der TLS-1.3-Verbindung des Webservers .	265
Abbildung A-11:	Erfolgreiche Validierung der Genesis-Datei des Webservers	266

Abbildung A-12: Erfolgreiche Validierung der Issuer Agent ACA-Py Swagger API . . .	269
Abbildung A-13: Darstellung der Netzwerkisolation innerhalb der Gesamtarchitektur	271
Abbildung A-14: Erfolgreiche Validierung der Netzwerkisolation	272

Listingsverzeichnis

Listing 1:	Zweite Iteration - Wallet DID Abfrage vor Out-of-Band Invitation	75
Listing 2:	Zweite Iteration - Out-of-Band Invitation	75
Listing 3:	Zweite Iteration - Wallet DID Abfrage nach Out-of-Band Invitation	76
Listing A-1:	Strukturierte Suchanfrage mit Booleschen Operatoren	112
Listing A-2:	Zertifikaterstellungsworkflow	209
Listing A-3:	Dockerfile - Sidecar-Proxy nginx	212
Listing A-4:	nginx_holder.conf	216
Listing A-5:	docker-compose.yml: DLT-Infrastruktur	218
Listing A-6:	docker-compose.yml: Revocation Registry	223
Listing A-7:	Dockerfile - acapy-base	225
Listing A-8:	docker-compose.yml: SSI-Agenten	229
Listing A-9:	Docker Compose Start der Gesamtarchitektur	234
Listing A-10:	von-network manage Skript	235
Listing A-11:	indy-tails-server manage Skript	254
Listing A-12:	Issuer Agent Boot Logs	267
Listing A-13:	pqc_didpeer4_fm - liboqs_wrapper.py	273
Listing A-14:	pqc_didpeer4_fm - pqc_peer4_creator.py	277
Listing A-15:	pqc_didpeer4_fm - pqc_peer4_resolver.py	279
Listing A-16:	pqc_didpeer4_fm - pqc_multicodec.py	281
Listing A-17:	pqc_didpeer4_fm - pqc_multikey.py	282
Listing A-18:	pqc_didpeer4_fm - pqc_didcomm_v1.py	284
Listing A-19:	pqc_didpeer4_fm - monkey_patches.py	293
Listing A-20:	pqc_didpeer4_fm - base_manager_patch.py	296
Listing A-21:	pqc_didpeer4_fm - askar_pqc_patch.py	310
Listing A-22:	pqc_didpeer4_fm - wallet_patch.py	328
Listing A-23:	pqc_didpeer4_fm - connection_target_patch.py	330
Listing A-24:	pqc_didpeer4_fm - validator_patch.py	333
Listing A-25:	pqc_didpeer4_fm - key_types.py	335
Listing A-26:	pqc_didpeer4_fm - key_type_patches.py	336
Listing A-27:	pqc_didpeer4_fm - multicodec_patch.py	339
Listing A-28:	pqc_didpeer4_fm - setup.py	341
Listing A-29:	pqc_didpeer4_fm - README.md	342
Listing A-30:	Dockerfile - acapy-base-pqc	346
Listing A-31:	multicodec_patch.py	350
Listing A-32:	Issuer Agent mit PQC-Plugin Boot Log	356

Listing A-33: Jupyter Notebook Cell 1	360
Listing A-34: Jupyter Notebook Cell 1 Output	363
Listing A-35: Jupyter Notebook Cell 2	363
Listing A-36: Jupyter Notebook Cell 2 Output	366
Listing A-37: Jupyter Notebook Cell 3	370
Listing A-38: Jupyter Notebook Cell 3 Output	371
Listing A-39: Jupyter Notebook Cell 4	371
Listing A-40: Jupyter Notebook Cell 4 Output	374
Listing A-41: Jupyter Notebook Cell 5	377
Listing A-42: Jupyter Notebook Cell 5 Output	377
Listing A-43: Jupyter Notebook Cell 6	378
Listing A-44: Jupyter Notebook Cell 6 Output	380
Listing A-45: Jupyter Notebook Cell 7	382
Listing A-46: Jupyter Notebook Cell 7 Output	384
Listing A-47: Jupyter Notebook Cell 8	394
Listing A-48: Jupyter Notebook Cell 8 Output	400
Listing A-49: Jupyter Notebook Cell 9	401
Listing A-50: Jupyter Notebook Cell 9 Output	406
Listing A-51: Jupyter Notebook Cell 10	407
Listing A-52: Jupyter Notebook Cell 10 Output	412
Listing A-53: Jupyter Notebook Cell 11	413
Listing A-54: Jupyter Notebook Cell 11 Output	414
Listing A-55: Jupyter Notebook Cell 12	414
Listing A-56: Jupyter Notebook Cell 12 Output	415
Listing A-57: Jupyter Notebook Cell 13	419
Listing A-58: Jupyter Notebook Cell 13 Output	422
Listing A-59: Jupyter Notebook Cell 14	423
Listing A-60: Jupyter Notebook Cell 14 Output	424
Listing A-61: Jupyter Notebook Cell 15	425
Listing A-62: Jupyter Notebook Cell 15 Output	426
Listing A-63: Jupyter Notebook Cell 16	427
Listing A-64: Jupyter Notebook Cell 16 Output	429
Listing A-65: Jupyter Notebook Cell 17	429
Listing A-66: Jupyter Notebook Cell 17 Output	433
Listing A-67: Jupyter Notebook Cell 18	434
Listing A-68: Jupyter Notebook Cell 18 Output	437
Listing A-69: Jupyter Notebook Cell 19	438

Listing A-70: Jupyter Notebook Cell 19 Output	439
Listing A-71: Jupyter Notebook Cell 20	439
Listing A-72: Jupyter Notebook Cell 20 Output	440
Listing A-73: Jupyter Notebook Cell 21	441
Listing A-74: Jupyter Notebook Cell 21 Output	443
Listing A-75: Jupyter Notebook Cell 14 Output Revocation	445
Listing A-76: Jupyter Notebook Cell 15 Output Revocation	445
Listing A-77: Jupyter Notebook Cell 16 Output Revocation	445
Listing A-78: Jupyter Notebook Cell 17 Output Revocation	446
Listing A-79: Jupyter Notebook Cell 18 Output Revocation	447
Listing A-80: Jupyter Notebook Cell 22	448
Listing A-81: Jupyter Notebook Cell 22 Output	450
Listing A-82: Jupyter Notebook Cell 23	451
Listing A-83: Jupyter Notebook Cell 23 Output	453
Listing A-84: Issuer acapy agent logs	459
Listing A-85: Issuer Sidecar-Proxy logs	460
Listing A-86: Jupyter Notebook Cell 9 Demonstration Kryptoagilität ed25519	461
Listing A-87: Jupyter Notebook Cell 9 Demonstration Kryptoagilität ed25519 Output	466
Listing A-88: Jupyter Notebook Cell 11 Demonstration Kryptoagilität ed25519	467
Listing A-89: Jupyter Notebook Cell 11 Demonstration Kryptoagilität ed25519 Output	467

Tabellenverzeichnis

Tabelle 1:	Übersicht ausgewählter PQC-Algorithmus-Kategorien	18
Tabelle 2:	Eigenschaften der Kryptoagilität	22
Tabelle 3:	Evaluationsepisoden nach dem FEDS-Framework	32
Tabelle 4:	Funktionale Anforderungen an SSI-Systeme	36
Tabelle 5:	Compliance Anforderungen an SSI-Systeme im KRITIS-Kontext	37
Tabelle A-1:	Zehn Prinzipien von Self-Sovereign Identity	104
Tabelle A-2:	Ausgewählte Methoden der PRISMA 2020 Richtlinien	106
Tabelle A-3:	Ein- und Ausschlusskriterien für die systematische Literaturrecherche	107
Tabelle A-4:	Überblick über die Entwicklung der Suchstrategie	109
Tabelle A-5:	Abgrenzung zentraler Schlüsselkonzepte	110
Tabelle A-6:	Keywords der Schlüsselkonzepte	111
Tabelle A-7:	Erste Iteration - Übersicht der relevanten Quellen	118
Tabelle A-8:	Erste Iteration - Relevanzbewertung der identifizierten Quellen	119
Tabelle A-9:	Ein- und Ausschlusskriterien für die systematische Literaturrecherche	168
Tabelle A-10:	Zweite Iteration - Übersicht der relevanten Quellen	174
Tabelle A-11:	Zweite Iteration - Relevanzbewertung der identifizierten Quellen	175
Tabelle A-12:	KI-Hilfsmittelverzeichnis	481

Abkürzungsverzeichnis

ACA-Py	Aries Cloud Agent Python
BO	Business Object
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certificate Authority
CISA	Cybersecurity and Infrastructure Security Agency
DID	Decentralized Identifiers
DLT	Decentralized Ledger Technology
DPKI	Decentralized Public Key Infrastructure
DSGVO	Datenschutz-Grundverordnung
DSR	Design Science Research
DSRM	Design Science Research Methodology
EBT	Enduring Business Theme
ECC	Elliptic Curve Cryptography
FEDS	Framework for Evaluation in Design Science
IAM	Identity Access Management
IO	Industrial Object
IoT	Internet of Things
IT	Information Technology
KRITIS	Kritische Infrastrukturen
NIST	National Institute of Standards and Technology
OQS	Open Quantum Safe
OT	Operational Technology
PKI	Public Key Infrastructure
PQC	Post-Quantum Cryptography
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
RFC	Request for Comments
RSA	Rivest-Shamir-Adleman
SSI	Self-Sovereign Identity
TLS	Transport Layer Security
VC	Verifiable Credentials
VDR	Verifiable Data Registry
VP	Verifiable Presentations
ZKP	Zero Knowledge Proof

Glossar

Cryptographically Relevant Quantum Computing (CRQC) beschreibt das Stadium, in dem Quantencomputer aktuelle Kryptosysteme brechen können (Geremew & Mohammad, 2024, S. 338). 2

Data-At-Rest bezieht sich auf Daten, die sich auf einem Computer, einem Server oder irgendwo in der Cloud befinden (Swanzy, Abukari & Ansong, 2024, S. 62) . 55, 58, 77, 92

Data-In-Motion bezieht sich auf Daten, die aktiv von einem Ort zu einem anderen über ein Netzwerk oder von einem lokalen Speichergerät zu einem Cloud-Speicher übertragen werden (Swanzy, Abukari & Ansong, 2024, S. 62) . 55, 58, 77, 92

DIDComm bezeichnet nach Badertscher, Banfi und Diaz (2024, S. 4732) ein kryptografisches Framework zur Etablierung sicherer Kommunikation zwischen Entitäten mit Decentralized Identifiers, das Sender-Anonymität und Authentizität kombiniert. DIDComm v1 spezifiziert ausschließlich X25519 für Key Exchange und Ed25519 für digitale Signaturen ohne Algorithmen-Aushandlung, wodurch keine native Kryptoagilität vorliegt (Badertscher, Banfi & Diaz, 2024, S. 4733, 4735) . 31, 32, 41, 43, 55, 58, 61, 63, 65, 66, 67, 69, 72, 74, 77, 85, 87, 92, 94, 95, 96, 97, 204, 227, 228

EBSCO ist eine umfassende, multidisziplinäre Datenbank, die eine breite Palette von akademischen Zeitschriften, Magazinen und anderen Publikationen abdeckt. Sie bietet Zugang zu einer Vielzahl von Fachgebieten, einschließlich Informatik, Cybersicherheit und Politikwissenschaften. Dies ist besonders wertvoll für das Thema, da es die Schnittstelle zwischen Technologie und nationaler Sicherheit betrifft („About EBSCO Information Services“, n. d.). 24, 114, 116, 170, 172

Hybride Schemata kombinieren zwei oder mehr Algorithmen derselben Kategorie, so dass das Gesamtsystem seine Sicherheit behält, solange mindestens eine der verwendeten Komponenten als sicher gilt (Bindel et al., 2019, S. 3) . 40, 46, 47, 92

Monkey-Patching bezeichnet eine Technik im Software-Engineering, bei der zur Laufzeit bestehende Klassen oder Methoden dynamisch verändert oder erweitert werden, ohne den ursprünglichen Quellcode zu modifizieren (Luo et al., 2025, S. 2) . 61, 63, 67, 72, 77, 78, 92, 95, 97, 99

Sidecar-Proxy ist ein zusätzlicher Microservice, der neben Business-Microservices in einem Pod bereitgestellt wird, um Proxy-Funktionen bereitzustellen, welche sämtliche ein- und ausgehende Kommunikation eines Pods abfangen und kryptografisch schützen, ohne die zugrundeliegende Anwendungslogik zu modifizieren (Berlato et al., 2024, S. 1–2) . 32, 39, 40, 44, 46, 47, 48, 49, 53, 92, 95, 96, 99, 100, 205, 227, 228

SzA Systeme zur Angriffserkennung sind im Sinne des Bundesministerium der Justiz (2009, § 2 Absatz 9b) durch technische Werkzeuge und organisatorische Einbindung unterstützte Prozesse zur Erkennung von Angriffen auf informationstechnische Systeme. Die Angriffserkennung erfolgt dabei durch Abgleich der in einem informationstechnischen System verarbeiteten Daten mit Informationen und technischen Mustern, die auf Angriffe hindeuten. 37, 99

1 Einleitung

1.1 Problemstellung und Motivation

Kritische Infrastrukturen (KRITIS) bilden das Rückgrat moderner Gesellschaften und sind essenziell für die nationale Sicherheit, wirtschaftliche Stabilität sowie die öffentliche Gesundheit und Sicherheit (Alcaraz & Zeadally, 2015, S. 53). Die fortschreitende Digitalisierung und Vernetzung dieser Systeme führt jedoch zu einer signifikanten Erweiterung der Angriffsfläche, wodurch KRITIS zunehmend komplexen Cyberangriffen ausgesetzt sind (Nosouhi et al., 2024, S. 1). Diese Entwicklung wird durch die Konvergenz von Information Technology (IT) und Operational Technology (OT) verstärkt, die traditionell isolierte industrielle Steuerungssysteme mit Unternehmensnetzwerken verbindet und neue Sicherheitsrisiken schafft (Oqliak et al., 2023, S. 1).

Die zunehmende Online-Verarbeitung und -Speicherung von Daten verschieben den Security Perimeter hin zur Identität, deren Verwaltung zum zentralen Aspekt der Cybersicherheit wird (Jøsang, 2025, S. 191). Identity Management nimmt in diesem Kontext eine zentrale Stellung ein, da es die Identifikation, Verifikation und Autorisierung von Personen sowie die Verwaltung von Zugriffsrechten auf kritische Ressourcen ermöglicht (Fritsch, 2020, S. 61). Identity Management stellt somit eine KRITIS dar, die vielen anderen gesellschaftlich relevanten KRITIS und Funktionen zugrunde liegt (Fritsch, 2020, S. 61). Die besondere Brisanz ergibt sich daraus, dass kompromittierte digitale Identitäten nicht nur den Zugang zu kritischen Systemen ermöglichen, sondern auch als Ausgangspunkt für weitreichende Angriffe dienen können (Nosouhi et al., 2024, S. 3). Analysen zeigen, dass Identity Access Management (IAM)-basierte Mitigationsansätze 42% der Angriffsvektoren beim „Initial Access“ und 86% der Angriffsvektoren beim „Lateral Movement“ effektiv adressieren können (Nosouhi et al., 2024, S. 3–4).

Trotz dieser Bedeutung arbeiten herkömmliche Identity Management Systeme überwiegend mit zentralisierten Frameworks, die inhärente Schwachstellen aufweisen. Zu diesen zählen Single Points of Failure, eingeschränkte Skalierbarkeit und begrenzte Datenschutzfunktionen (Ramírez-Gordillo et al., 2025, S. 1). In solchen Modellen besitzen Nutzer keine effektive Kontrolle über ihre digitalen Identitäten, da Authentifikatoren, Daten und Credentials in Datenbanken Dritter gespeichert werden, die letztlich die Hoheit ausüben (Allende López, 2020, S. 15). Dies führt zu erheblichen Datenschutzrisiken und schränkt die Nutzerautonomie ein, da Rechte wie Einwilligung, Löschung („Recht auf Vergessenwerden“), Portabilität und Pseudonymisierung kaum technisch garantiert werden können (Allende López, 2020, S. 18–19).

Das Konzept der Self-Sovereign Identity (SSI) adressiert diese architektonischen Limitierungen durch einen fundamentalen Paradigmenwechsel. Die Souveränität über die Identitätsverwaltung wird direkt zum Nutzer verlagert, der seine digitalen Assets und Nachweise mithilfe von lokalen Wallets autonom kontrolliert (Allende López, 2020, S. 7). Technologisch wird dieser Ansatz durch die Integration von Decentralized Identifiers (DID) und Decentralized Ledger Technology (DLT) realisiert, die als dezentrale Vertrauensanker fungieren und dadurch die Abhängigkeit von zentralen Zertifizierungsstellen auflösen (Ramírez-Gordillo et al., 2025, S. 5–6). Ergänzend ermöglichen Verifiable Credentials (VC) und Verifiable Presentations (VP) einen kryptografisch gesicherten Austausch von Identitätsdaten, bei dem der Nutzer selektiv entscheiden kann, welche Informationen er wem präsentiert, ohne dabei die Datenhoheit abzugeben (Allende López, 2020, S. 71, 74, 77). Diese Architektur eliminiert durch die verteilte Validierung von Transaktionen systemisch zentrale Angriffsvektoren, was die Resilienz gegenüber Ausfällen und Manipulationen signifikant erhöht (Ramírez-Gordillo et al., 2025, S. 32).

Während SSI die architektonischen Schwachstellen adressiert, entwickeln sich die rasanten Fortschritte im Quantum Computing parallel zu einer tiefgreifenden Bedrohung für traditionelle kryptografische Systeme (Mamatha, Dimri & Sinha, 2024, S. 1). Diese Bedrohung ergibt sich aus der Fähigkeit von Quantencomputern, spezifische mathematische Probleme schneller zu lösen als klassische Computer (Geremew & Mohammad, 2024, S. 339). Nach Einschätzung der Cybersecurity and Infrastructure Security Agency (CISA) könnte die kritische Phase dieser Entwicklung bereits um 2030 mit dem Erreichen von Cryptographically Relevant Quantum Computing (CRQC) eintreten (Geremew & Mohammad, 2024, S. 344).

Zwei bedeutende Quantenalgorithmen sind hierbei von besonderer Relevanz (Geremew & Mohammad, 2024, S. 341). Der von Shor (1994, S. 1) entwickelte Shor-Algorithmus löst Probleme der ganzzahligen Faktorisierung und des diskreten Logarithmus effizient und bedroht somit die Sicherheit vieler asymmetrischer Kryptosysteme wie Rivest-Shamir-Adleman (RSA) und Elliptic Curve Cryptography (ECC) (Geremew & Mohammad, 2024, S. 341). Der von Grover (1996, S. 1) entwickelte Grover-Algorithmus bietet eine quadratische Beschleunigung für die Suche in unstrukturierten Datenbanken und verringert die Sicherheit symmetrischer Kryptografie durch Halbierung der effektiven Schlüsselgröße (Bernstein, 2010, S. 1).

Die Bedrohung durch Quantencomputer manifestiert sich besonders in der sogenannten „Harvest Now, Decrypt Later“-Strategie, bei der Angreifer verschlüsselte Daten bereits heute abfangen und speichern, um sie später mit Quantencomputern zu entschlüsseln (Geremew & Mohammad, 2024, S. 344–345). Diese Angriffsstrategie gefährdet insbeson-

dere Daten mit langfristiger Schutzbedürftigkeit, wie sie in KRITIS regelmäßig verarbeitet werden (Geremew & Mohammad, 2024, S. 338–339, 345). Klassische kryptografische Systeme, die derzeit IAM-Infrastrukturen absichern, werden somit mittelfristig ihre Schutzwirkung verlieren (Geremew & Mohammad, 2024, S. 339, 341, 351).

Als Antwort darauf entwickelt die Kryptografie-Community quantenresistente Verfahren, die als Post-Quantum Cryptography (PQC) zusammengefasst werden (Mamatha, Dimri & Sinha, 2024, S. 1). Das National Institute of Standards and Technology (NIST) finalisierte im August 2024 die ersten drei PQC-Standards (ML-KEM, ML-DSA, SLH-DSA), die auf alternativen mathematischen Problemen wie Gitterstrukturen basieren (National Institute of Standards and Technology, 2024a, 2024b, 2024c) und in der Umstellung von Internet Protokollen eingesetzt werden sollen (Solavagione & Vesco, 2025). In diesem Kontext offenbart sich jedoch eine kritische Sicherheitslücke bei dezentralen Identitäten, da sich aktuelle SSI-Implementierungen auf klassische kryptografische Primitive stützen, die durch künftige Quantencomputer kompromittiert werden können (Solavagione & Vesco, 2025, S. 2–3). Um die strukturellen Resilienzvorteile der Dezentralisierung auch langfristig für KRITIS zu erhalten, ist eine Migration auf PQC erforderlich.

1.2 Stand der Forschung und Identifikation der Forschungslücke

Die systematische Literaturrecherche (Kapitel 3.1, Anhang 2) zum gegenwärtigen Stand der Forschung wurde nach ausgewählten Methoden des Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 Standards (Page et al., 2021, S. 1) durchgeführt und in zwei iterativen Phasen organisiert. Dieser Ansatz ermöglicht es, sowohl die initiale Problemidentifikation als auch die Aktualisierung der Wissensbasis unter Berücksichtigung des dynamischen Charakters des Forschungsfeldes abzubilden. Folgend die Synthese der Ergebnisse der beiden Iterationen der Literaturrecherche (Tabelle A-7 und Tabelle A-10).

Im Bereich der dezentralisierten Identitätsverwaltung existiert eine wachsende Forschungstätigkeit, die technologische Innovationen mit regulatorischen Anforderungen verbindet. Die Arbeiten in diesem Feld lassen sich in vier zentrale Themencluster einordnen, die jeweils unterschiedliche Aspekte der SSI-Technologie und ihrer sicherheitstechnischen Anforderungen für KRITIS adressieren.

Der erste Cluster umfasst grundlegende Arbeiten zu blockchain-basierten Identitätsmanagementsystemen. Feng et al. (2025, S. 11) zeigt auf, dass Blockchain-Technologie neue Paradigmen für dezentrale Identitätsverwaltung eröffnet, in denen Nutzer die Kontrolle

über ihre Identitätsdaten zurückgewinnen und ihre Abhängigkeit von zentralisierten Instanzen reduzieren können. Der historische Entwicklungspfad führt von traditionellen zentralisierten und föderalen Architekturmodellen hin zu blockchain-basierten dezentralisierten Lösungen, die verteilte Ledger-Technologien als Vertrauensanker einsetzen (Feng et al., 2025, S. 11). Allerdings dokumentiert die umfassende Analyse unzureichend adressierte Sicherheitsherausforderungen wie Interoperabilität, Revocation-Management und Quantenresilienz (Feng et al., 2025, S. 2). Diese Lücken in der aktuellen Forschung sind besonders kritisch, da sie die praktische Deploybarkeit dezentraler Identitätslösungen in regulativ kontrollierten Sektoren einschränken.

Der zweite Cluster adressiert die Quantum-Computing-Bedrohung und die Standardisierung von PQC. Barrett-danes und Ahmad (2025, S. 2–3) dokumentieren, dass trotz rascher Fortschritte bei PQC-Lösungen erhebliche Lücken zwischen theoretischer Robustheit und praktischer Implementierbarkeit bestehen. Barrett-danes und Ahmad (2025, S. 2–3) weisen explizit auf die NIST-Standardisierung von vier PQC-Algorithmen im Jahr 2024 hin, vermerken jedoch, dass zwei vorangegangene Kandidaten während des Testens kompromittiert wurden und selbst die approbierten Algorithmen weitere praktische Validierung und Optimierung für ressourcenbegrenzte Umgebungen benötigen. Insbesondere im Kontext von Internet of Things (IoT)-Geräten zeigen sich erhebliche Skalierungsprobleme und Integrationskomplexität (Barrett-danes & Ahmad, 2025, S. 2–3). Die Autoren identifizieren ein kritisches Ungleichgewicht. Während 81% der analysierten Literatur einen dominanten Fokus auf die Entwicklung, Bewertung und Optimierung von kryptografischen PQC-Frameworks setzen, adressieren nur 4,8% der Studien tatsächliche Implementierungsbarrieren (Barrett-danes & Ahmad, 2025, S. 14–15). Dieses starke Ungleichgewicht in der Forschungslandschaft weist auf eine fundamentale Forschungslücke zwischen akademischer Lösung und praktischer Machbarkeit hin. Komplementär dazu erörtert Szymanski (2024, S. 1) eine neuartige Software-Defined-Deterministic-IIoT-Architektur mit quantensicheren Verschlüsselungsverfahren für KRITIS. Der Ansatz kombiniert deterministische Netzwerkmechanismen mit Field-Programmable-Gate-Array-basierter Hardwareverstärkung, um DoS/DDoS-Angriffe vollständig zu eliminieren und Zero-Trust-Architekturen hardwaregestützt umzusetzen (Szymanski, 2024, S. 1–2). Das Konzept der Authenticated Encrypted Deterministic Channels zeigt, dass Architektur-Design und Kryptografie eng miteinander verflochten sein müssen, um KRITIS angemessen zu schützen.

Der dritte Cluster befasst sich mit hybriden Ansätzen und fortgeschrittenen Signaturschemata für ressourcenlimitierte Umgebungen. Nouma und Yavuz (2024, S. 2) präsentieren Hardware-Assisted Efficient Signatures, die Forward-Secure PQC-Signaturen mit hoher Signer-Efficiency kombinieren. Die Autoren zeigen, dass NIST-PQC-Standards für embedded IoT-Komponenten faktisch ungeeignet sind (Nouma & Yavuz, 2024, S. 2–3). Dies wird

durch Arbeiten von Alam, Hoffstein und Cambou (2024, S. 1–2) ergänzt, die Challenge-Response-Mechanismen für dezentrale PQC-Keygenerierung demonstrieren und damit einen Brückenschlag zwischen dezentraler Kontrolle und zentraler Verifikation ermöglichen. Der Einsatz von CRYSTALS-Dilithium als standardisierten PQC-Algorithmus zeigt, dass gitterbasierte Verfahren auch für verteilte Schlüsselerzeugung praktikabel sind (Alam, Hoffstein & Cambou, 2024, S. 17).

Der vierte Cluster behandelt regulatorische und standardisierungstechnische Anforderungen an dezentrale Identitätssysteme. Sharif et al. (2022, S. 23–24) zeigen, dass die europäische eIDAS-2.0-Regulierung einen Paradigmenwechsel von zentralisierten zu selbst-bestimmten Identitätsmodellen vorsieht, ohne dabei konkrete technische Anforderungen an Quantenresilienz in dezentralen Wallets zu spezifizieren. Ähnlich dokumentiert die internationale Regulatory-Landscape, dass Cybersicherheitsstandards für KRITIS zwar zunehmend PQC mandatieren, dabei aber Implementierungspfade und konkrete Integrationsvorgaben für dezentrale Systeme noch nicht ausreichend entwickelt haben (Radanliev, 2023, S. 105–106).

Zusammenfassend offenbart sich eine Forschungslücke an der Schnittstelle zwischen dezentraler Identitätsverwaltung, PQC und der Sicherheit kritischer Infrastrukturen. Während sowohl SSI-Technologien als auch PQC-Standards als Einzeltechnologien einen ausgereiften Standardisierungsgrad erreicht haben, existiert bislang praktisch keine systematische Forschungsarbeit zur kombinierten Evaluation und zum sicherheitsfokussierten Design eines quantensicheren SSI-Prototypen, der gleichzeitig KRITIS-spezifische Anforderungen und regulatorische Compliance erfüllt.

1.3 Zielsetzung und Forschungsfragen

Die vorliegende Masterarbeit adressiert die in Kapitel 1.2 identifizierte Forschungslücke an der Schnittstelle zwischen dezentraler Identitätsverwaltung, PQC und KRITIS-Sicherheit. Die übergeordnete Zielsetzung ist die Entwicklung und empirische Evaluation eines funktionsfähigen blockchain-basierten SSI-Prototypen mit integrierter PQC im Kontext von KRITIS. Im Kern wird untersucht, ob und unter welchen architektonischen Bedingungen NIST-standardisierte PQC-Algorithmen nicht-invasiv in bestehende SSI-Frameworks integriert werden können, ohne sowohl die Dezentralisierungsprinzipien der SSI-Technologie als auch die regulatorischen Anforderungen von KRITIS zu gefährden. Die zentrale Herausforderung besteht darin, ob Quantensicherheit konsistent über mehrere architektonische Schichten hinweg operationalisierbar ist und damit eine praktikable Synthese der drei

Anforderungsdimensionen Dezentralisierung, Quantensicherheit und KRITIS-Compliance gelingen kann.

Aus dieser übergeordneten Zielsetzung ergeben sich drei konkrete und operationalisierbare Forschungsfragen, die die „Technical Feasibility“ einer quantensicheren SSI-Architektur für KRITIS auf unterschiedlichen Ebenen adressieren:

FF1 – Systemarchitektur & Compliance: Wie kann ein blockchain-basiertes SSI-System unter Einsatz von PQC gestaltet werden, um die regulatorischen und technischen Anforderungen von KRITIS nachhaltig zu erfüllen?

FF1 operationalisiert die Frage auf architektonischer Ebene, indem sie das Design der Gesamtstruktur adressiert und darauf abzielt, architektonischen Muster, Schichten und Integrationspunkte zu ermitteln, um Quantensicherheit und regulatorische Compliance simultan zu realisieren.

FF2 – Algorithmenauswahl & Sicherheitsbewertung: Welche PQC-Algorithmen eignen sich für die Integration in SSI-Systeme hinsichtlich Sicherheit und Interoperabilität, insbesondere im Kontext von KRITIS?

FF2 konkretisiert die Frage auf algorithmischer Ebene, indem sie die Wahl konkreter kryptografischer Standards adressiert und untersucht, welche Algorithmen und Parameter unter realistischen Integrationsbedingungen praktikabel und sicher sind.

FF3 – Kryptografische Agilität: Welche kryptografischen Agilitätsmechanismen sind erforderlich, um zukünftige PQC-Algorithmenupdates ohne Systemunterbrechung zu ermöglichen?

FF3 adressiert die Frage auf der Ebene der Systemevolution, indem sie untersucht, welche Mechanismen in die Architektur eingebaut werden müssen, um langfristige Wartbarkeit und Anpassungsfähigkeit an zukünftige kryptografische Standards zu sichern.

1.4 Methodisches Vorgehen

Die vorliegende Masterarbeit folgt dem Design Science Research (DSR)-Paradigma (Hevner et al., 2004, S. 75), das auf die Erschaffung und wissenschaftliche Evaluation innovativer Artefakte ausgerichtet ist (Kapitel 3.2). Das methodische Vorgehen wird durch drei zentrale Komponenten strukturiert. Als Fundament dient eine systematische Literaturrecherche (Kapitel 3.1) nach PRISMA 2020 Standards (Page et al., 2021, S. 1) in zwei zeitlich versetzten Phasen, die sowohl die initiale Problemidentifikation als auch die Wissensaktualisierung unter Berücksichtigung des dynamischen Forschungsfeldes gewährleistet. Dies sichert Transparenz, Reproduzierbarkeit und wissenschaftliche Nachvollziehbarkeit.

Kernaktivität ist die iterative Artefaktentwicklung nach dem Design Science Research Methodology (DSRM) Prozessmodell (Peffers et al., 2007, S. 46, 54), welches einen strukturierten sechsphasigen Rahmen bereitstellt (Kapitel 3.4). Der Research Entry erfolgt als Problem-Centered Initiation, motiviert durch spezifische technologische Bedrohungen und regulatorische Anforderungen. Das Entwicklungsverfahren organisiert sich als zweizyklischer Iterationsprozess, wobei jeder Zyklus Lösungsdefinition, Designentwicklung und formative Evaluation umfasst. Diese Struktur ermöglicht systematisch, Erkenntnisse aus einer Implementierungsrounde zur iterativen Verbesserung der nachfolgenden Phase einzubeziehen.

Die Evaluation folgt dem Framework for Evaluation in Design Science (FEDS) (Venable, Pries-Heje & Baskerville, 2016, S. 77). Gewählt wurde eine Technical Risk Efficacy Strategy, die mit formativen künstlichen Tests beginnt und in summativer Evaluation endet (Kapitel 3.3). Da das primäre Entwicklungsrisiko technischer Natur ist, erfolgen Tests in einer kontrollierten Entwicklungsumgebung (Anhang 3.2.1) bevor Feldtests aus Sicherheitsgründen vertretbar wären. Die Evaluation untergliedert sich in diskrete Episoden parallel zu den Entwicklungszyklen und überprüft einzelne Komponenten und Subsysteme unter kontrollierten Bedingungen.

Die Operationalisierung der Forschungsfragen erfolgt bidirektional. Auf normativer Ebene werden explizite Designziele pro Iterationsphase (Kapitel 4.1 und 5.1) etabliert, die sich unmittelbar aus den übergeordneten Forschungsfragen ableiten und als Vorgaben für Architektur- und Implementierungsentscheidungen fungieren. Auf operativer Ebene erfolgt eine systematische Konkretisierung in funktionalen Anforderungen (Kapitel 4.2.1), die den System-Lebenszyklus abdecken, sowie in spezifischen Compliance-Anforderungen (Kapitel 4.2.2) aus sicherheitsrelevanten Standards und Richtlinien. Diese strukturierte, bidirektionale Operationalisierung garantiert wissenschaftliche Nachvollziehbarkeit und Evidenzbasierung des gesamten methodischen Vorgehens.

1.5 Scope und Limitationen

Der begrenzte Rahmen einer Masterarbeit erfordert eine klare Fokussierung und Priorisierung der Forschungsaktivitäten. Die vorliegende Arbeit konzentriert sich daher auf die „Technical Feasibility“ und „Compliance Readiness“ eines blockchain-basierten SSI-Systems mit integrierter PQC für KRITIS. Diese Schwerpunktsetzung leitet sich direkt aus der in Kapitel 1.2 identifizierten Diskrepanz zwischen theoretischer PQC-Forschung und fehlender praktischer Implementierungsexpertise ab (Barrett-danes & Ahmad, 2025, S.

14–15). Während mathematische Primitive bereits standardisiert vorliegen (National Institute of Standards and Technology, 2024a, 2024b, 2024c), ist deren architektonische Integration in verteilte Identitätssysteme bisher kaum empirisch validiert. Das primäre Forschungsziel ist folglich der Nachweis, dass eine solche Integration unter Erhaltung der Systemintegrität prinzipiell möglich ist, bevor in zukünftigen Arbeiten eine Optimierung auf Effizienz erfolgen kann.

Der methodische Rahmen umfasst zwei DSR-Iterationen, die jeweils durch formative Evaluationen begleitet werden und in eine summative Gesamtevaluation münden. Diese Beschränkung auf zwei Entwicklungszyklen erlaubt eine sorgfältige Dokumentation und wissenschaftlich fundierte Reflexion der Designentscheidungen, limitiert jedoch naturgemäß die Anzahl möglicher Architekturvarianten und experimenteller Optimierungsschritte. Der Forschungsschwerpunkt liegt hierbei explizit auf der SSI-Agent-Schicht, während die zugrundeliegende DLT- und Revocations-Infrastruktur bewusst als reine Labor-Komponenten ohne Anspruch auf Produktivreife eingesetzt werden. Diese Entscheidung ist methodisch begründet, da der Erkenntnisfokus auf der Integration von Post-Quantum-Algorithmen in die Identitätsverwaltungsschicht liegt und nicht auf der Optimierung der Blockchain-Konsensverfahren oder der skalierten Revocation-Infrastruktur. Die gewählte Architektur ermöglicht reproduzierbare Experimente unter kontrollierten Bedingungen, erlaubt jedoch keine Aussagen über das Verhalten in produktiven, verteilten Netzwerken mit hunderten Validierungsknoten oder unter realen Lastbedingungen.

Die Anforderungsanalyse konzentriert sich auf funktionale Anforderungen, die den vollständigen SSI-Lebenszyklus abdecken, sowie auf technische Compliance-Anforderungen, die aus Richtlinien des Bundesamt für Sicherheit in der Informationstechnik (BSI), der Datenschutz-Grundverordnung (DSGVO) und internationalen Standards abgeleitet werden. Nichtfunktionale Anforderungen wie Performance, Durchsatz, Latenzen oder Ressourcenverbrauch werden explizit ausgeklammert. Diese Delimitation ist methodisch begründet dadurch, dass sich PQC-Algorithmen sowie deren Integration teilweise noch in frühen Standardisierungsphasen befinden, sodass Performance-Messungen an Referenzimplementierungen von begrenzter Aussagekraft für zukünftige produktive Systeme wären (Demir, Bilgin & Onbasli, 2025, S. 3–5). Zusätzlich erfolgt die Evaluation bewusst in einer artifiziellen Laborumgebung, um isolierte funktionale Nachweise ohne Störfaktoren realer Netzwerke zu erbringen, wodurch Rückschlüsse auf die Performance in produktiven Systemen methodisch nicht valide wären.

Die Evaluation der Compliance-Anforderungen erfolgt aus einer rein technisch-informatischen Perspektive und adressiert ausschließlich technologische Umsetzungsaspekte. Prozessuale und organisatorische Compliance-Dimensionen liegen außerhalb

des Untersuchungsrahmens. Dies entspricht der Zielsetzung, „Compliance Readiness“ als technische Grundlage nachzuweisen, nicht jedoch vollständige organisatorische Compliance zu gewährleisten, die umfassende Governance-Strukturen und operative Prozesse erfordert.

Die Arbeit untersucht die Integration von PQC in bestehende SSI-Frameworks, entwickelt jedoch keine neuen kryptografischen Primitive. Die Forschung adressiert somit die architektonische Integration und Anwendbarkeit standardisierter Algorithmen in dezentralen Identitätssystemen, nicht deren mathematische Sicherheit oder algorithmische Optimierung. Da sich sowohl die PQC-Standardisierungsprozesse als auch die regulatorischen Rahmenbedingungen für digitale Identitäten in kontinuierlicher Weiterentwicklung befinden, bildet diese Arbeit den Wissensstand zum Zeitpunkt der Erstellung ab und kann zukünftige Änderungen in Standards oder Regularien nicht antizipieren.

1.6 Aufbau der Arbeit

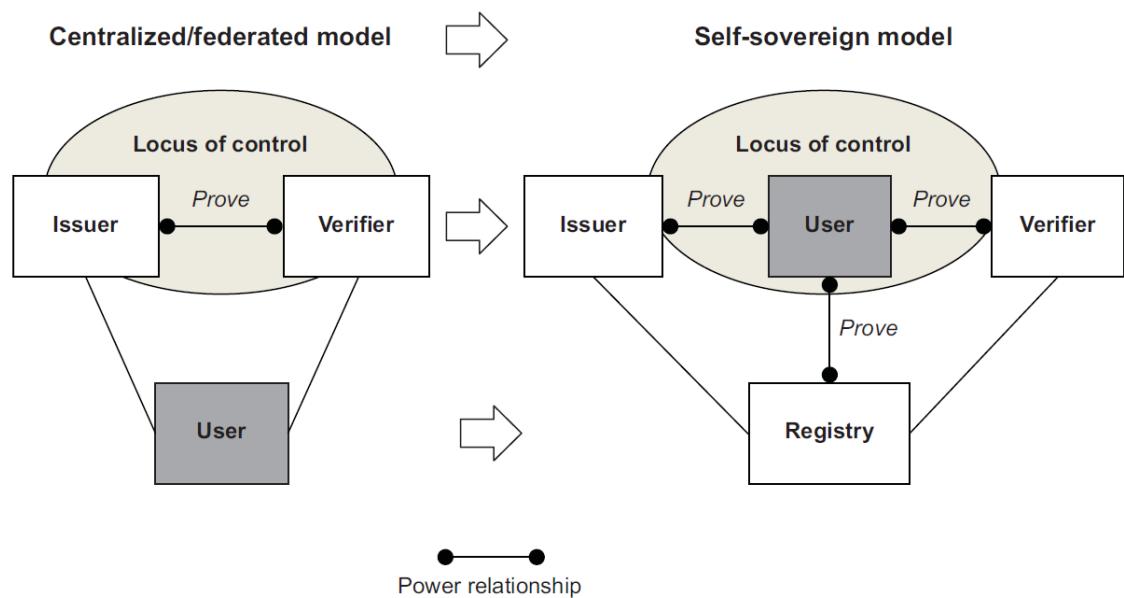
Die vorliegende Masterarbeit gliedert sich in acht Kapitel, die eine kohärente Argumentation von der Problemstellung über das Lösungsdesign bis zur Reflexion der Ergebnisse aufbauen. Nach der Einleitung behandelt Kapitel 2 die theoretischen Grundlagen zu SSI, Blockchain-Technologie, PQC sowie KRITIS, Compliance-Anforderungen und Kryptoagilität. Kapitel 3 dokumentiert das methodische Vorgehen bestehend aus systematischer Literaturrecherche nach PRISMA 2020 Standards, der DSR-Methodik und dem FEDS-Framework zur Evaluation. Die Kapitel 4 und Kapitel 5 präsentieren zwei iterative Zyklen der Artefaktentwicklung mit jeweiligen formativen Evaluationsepisoden. Kapitel 4 fokussiert auf die Basis-Implementierung der SSI-Architektur mit PQC auf Transport-Ebene, während Kapitel 5 eine Verfeinerung auf Applikations-Ebene durch Plugin-basierte Integration durchführt. Kapitel 6 führt eine summative Evaluation des finalen Artefakts durch und validiert funktionale Anforderungen, KRITIS-Compliance-Anforderungen sowie die erreichte Kryptoagilität. Kapitel 7 beantwortet die Forschungsfragen, reflektiert kritisch die Ergebnisse und artikuliert wissenschaftliche sowie praktische Beiträge. Kapitel 8 fasst die zentralen Ergebnisse zusammen und identifiziert zukünftige Forschungsrichtungen.

2 Theoretische Grundlagen

2.1 Self-Sovereign Identity

SSI ist ein dezentrales Identitätsparadigma, das Nutzern die vollständige Kontrolle über ihre persönlichen Daten und digitalen Identitäten ermöglicht (Satybaldy, Ferdous & Nowostawski, 2024, S. 1). Im Gegensatz zu traditionellen zentralisierten oder föderalen Identitätsmanagementsystemen, bei denen Identitätsanbieter als Intermediäre agieren, verleiht SSI dem Einzelnen die Autorität, eigenständig über die Verfügbarkeit und Weitergabe seiner digitalen Identitätsattribute zu bestimmen (Schardong & Custódio, 2022, S. 1). Diese Kontrollverschiebung nach Preukschat und Reed (2021, S. 12) vom Issuer und Verifier hin zum User wird in Abbildung 1 dargestellt.

Abbildung 1
Kontrollverschiebung vom Issuer und Verifier zum User



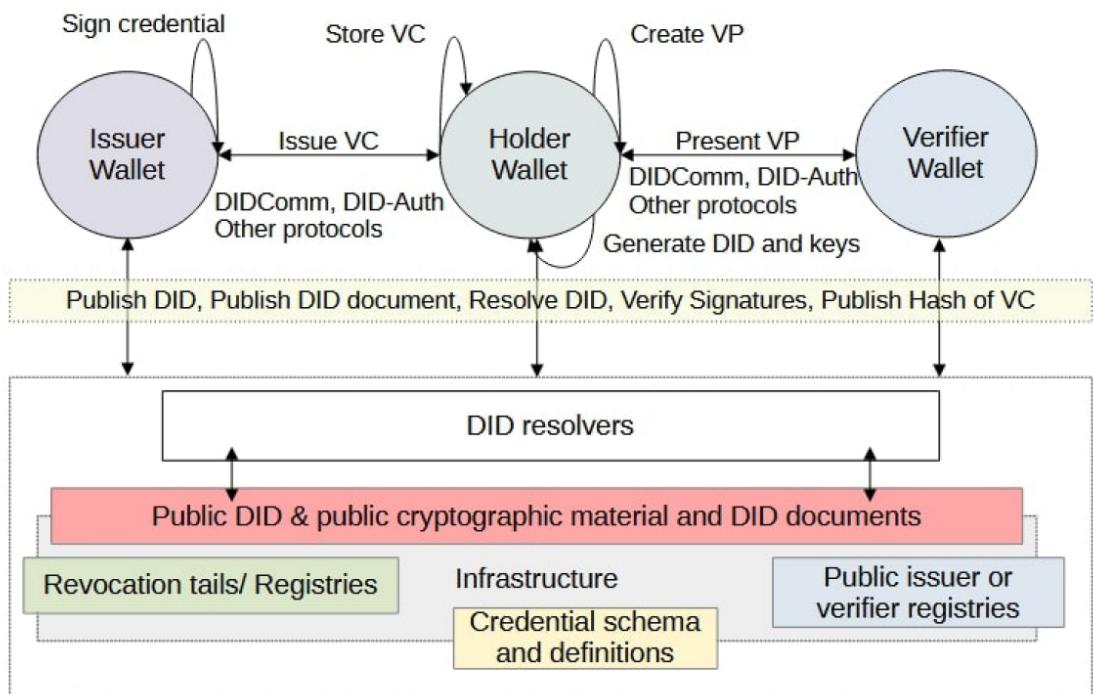
Anmerkung. Aus Preukschat & Reed (2021, S. 12).

Die Abgrenzung gegenüber anderen Identitätsmanagementsystemen wird durch die zehn Prinzipien von SSI Allen (2016) verdeutlicht, die in Tabelle A-1 aufgeführt sind. Diese Prinzipien waren als Reaktion auf die Nachteile bisheriger digitaler Identitätsmanagementsysteme entstanden, beschreiben keine spezifische technische Lösung, aber fungieren als konzeptionelle Anforderungen für die Umsetzung einer SSI-Lösung (Strüker et al., 2021, S. 12–13).

2.1.1 Kernkomponenten der SSI-Architektur

Ein funktionierendes SSI-System (Abbildung 2) besteht aus drei essentiellen Rollen, unterstützenden Technologiekomponenten und einer darunterliegenden dezentralen Infrastruktur (Naghmouchi & Laurent, 2025, S. 4). Die Interaktion zwischen autonomen Akteuren erfolgt durch eigenständig generierte Identifikatoren und kryptographisches Material, unterstützt durch Wallet-Applikationen (Naghmouchi & Laurent, 2025, S. 4).

Abbildung 2
SSI-Systemarchitektur



Anmerkung. Aus Naghmouchi & Laurent (2025, S. 4).

Der Issuer ist eine vertrauenswürdige Partei, dessen Identitätsinformationen und Public Key öffentlich einsehbar sind (Strüker et al., 2021, S. 20). Der Holder steht im Zentrum des SSI-Modells, fordert Credentials von Issuern an, speichert diese in einem digitalen Wallet und präsentiert bei Bedarf Nachweise an Verifier (Naghmouchi & Laurent, 2025, S. 4). Der Verifier fordert Nachweise an, prüft deren Validität und gewährt oder verweigert auf dieser Basis Zugang zu Diensten (Naghmouchi & Laurent, 2025, S. 4). Diese Rollen sind austauschbar, sodass ein Nutzer kontextabhängig zwischen ihnen wechseln kann (Naghmouchi & Laurent, 2025, S. 4).

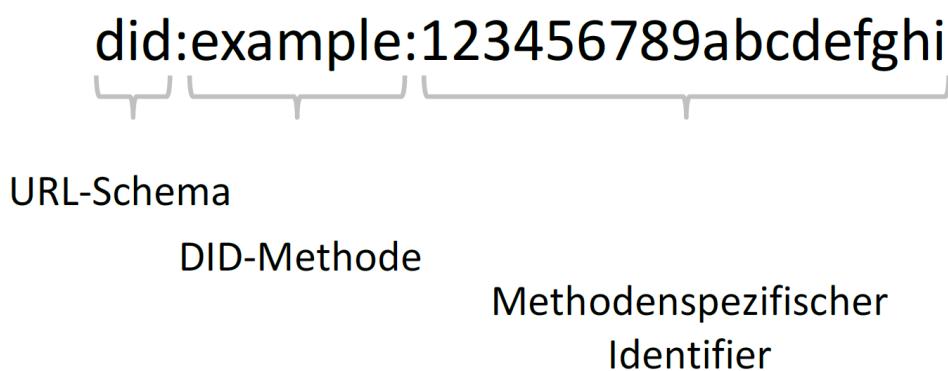
2.1.2 Infrastruktur

Die Infrastruktur-Schicht bildet die Basis von SSI-Systemen und dient als Decentralized Public Key Infrastructure (DPKI) zur Verwaltung von Public Keys, als Verifiable Data Registry (VDR) (Preukschat & Reed, 2021, S. 201) für öffentliche Identifikatoren und DID-Dokumente, sowie zur Veröffentlichung von Credential Schemas, Definitions und Revokationsinformationen (Naghmouchi & Laurent, 2025, S. 4–5, 7). Die Infrastruktur kann DLT, Blockchain-Technologie und/oder Web-basierte Technologie nutzen (Naghmouchi & Laurent, 2025, S. 8). Dezentrale Infrastrukturen werden bevorzugt eingesetzt, um die Prinzipien der Nutzerautonomie und -kontrolle zu erfüllen, da eine private Infrastruktur unter Kontrolle weniger Entitäten die Selbstsouveränität der Nutzer nicht gewährleisten kann (Naghmouchi & Laurent, 2025, S. 4).

2.1.3 Identifikatoren und kryptographisches Material

DID sind weltweit eindeutige, kryptographisch verifizierbare Identifikatoren, die dem Aufbau dargestellt in Abbildung 3 folgen und es verschiedenen Parteien ermöglichen, mehrere Identitäten zu kontrollieren, um ihre Privatsphäre zu schützen (Strüker et al., 2021, S. 22). In Verbindung mit kryptographischem Material wie Public Keys werden sie zur Identifikation von SSI-Nutzern verwendet, unabhängig von ihrer eingenommenen Rolle (Naghmouchi & Laurent, 2025, S. 5). Die Authentifizierung über Identifikatoren stellt die zwei grundlegenden Funktionalitäten des Identitätsmanagements (Identifikation und Authentifizierung) bereit (Naghmouchi & Laurent, 2025, S. 5).

Abbildung 3
Bestandteile eines DID



Anmerkung. Aus Naghmouchi & Laurent (2025, S. 4).

SSI nutzt Identifikatoren, die keine Registrierungsstellen oder Identity Provider erfordern, wobei die häufigsten Identifikatoren dem DID-Standard des W3C folgen (Naghmouchi & Laurent, 2025, S. 5). DID werden von ihren Eigentümern erstellt, sind mit einem Public Key verknüpft, der einem Private Key zugeordnet ist, sodass die Eigentümerschaft durch Challenge-Response-Verfahren verifiziert werden kann (Naghmouchi & Laurent, 2025, S. 5). Jede DID ist mit einem DID-Dokument verbunden, in welchem ein Service-Endpoint enthalten ist (Strüker et al., 2021, S. 23).

2.1.4 Credentials und Presentations

VC sind digital signierte Sammlungen von Attributen, die einem Standard unterliegen und kryptographisch verifizierbar sind (Strüker et al., 2021, S. 16–17). VC enthalten Revokationsmechanismen und kryptographisches Material, welches die Integrität des Credentials, die Identifizierung des Issuers und Nicht-Abstreitbarkeit gewährleistet (Schardong & Custódio, 2022, S. 5). Credentials enthalten einen Proof, der sie hinsichtlich Urheberschaft (Issuer-Seite) und Besitz (Holder-Seite) verifizierbar macht (Naghmouchi & Laurent, 2025, S. 5). Credentials werden üblicherweise mit Schlüsseln signiert, die mit vom Nutzer erstellten DID verknüpft sind (Naghmouchi & Laurent, 2025, S. 5).

Ein Credential ist mit einem Credential Schema verbunden, das Attribute, Formate und Typen beschreibt, während eine Credential Definition die Issuer-spezifische Instanz mit kryptographischen Signatur- und Verifikationsmethoden darstellt (Naghmouchi & Laurent, 2025, S. 5). Selective Disclosure definiert, dass der Holder nur spezifische Attribute aus dem Credential offenbaren muss, nicht das gesamte Credential (Strüker et al., 2021, S. 29). Die Präsentationsphase besteht darin, dass der Holder die angeforderten Attribute einem Verifier präsentiert. Dies geschieht entweder durch Vorlage vollständiger Credentials oder durch Selektion und Kombination einzelner Attribute aus verschiedenen Credentials mittels VP (Naghmouchi & Laurent, 2025, S. 5).

2.1.5 Wallet-Applikation (Edge Agent und Cloud Agent)

Kommunikationsprotokolle zwischen Peers, die Speicherung samt Austausch von VC und VP, sowie das Signieren und Verifizieren von Credentials werden über eine Wallet-Applikation abgewickelt (Naghmouchi & Laurent, 2025, S. 5). Digital Wallets ermöglichen Holdern ebenfalls die Generierung (Naghmouchi & Laurent, 2025, S. 5), die sichere Speicherung und Verwaltung von Private Keys und DID-Dokumenten (Strüker et al., 2021, S.

18), sowie deren Veröffentlichung auf der Infrastruktur, wodurch eine Befähigung zur Peer-to-Peer-Authentifizierung durch DID-Resolution stattfinden kann (Naghmouchi & Laurent, 2025, S. 5). Ein Wallet besteht typischerweise aus einem Cloud Agent, der als Online-Service in der Cloud läuft, und einem Edge Agent, der auf dem Gerät des Nutzers installiert ist (Naghmouchi & Laurent, 2025, S. 5).

2.1.6 Vertrauensmechanismus und Verifikationsprozess

Das Vertrauen in SSI-Systemen basiert nicht auf der Abhängigkeit von zentralisierten Intermediären, sondern auf kryptographischen und dezentralen Mechanismen. Wenn ein Holder eine VP an einen Verifier sendet, prüft dieser wer das zugrundeliegende Credential signiert hat, ob die Presentation korrekt konstruiert ist, und ob das Credential noch gültig ist und nicht revocated wurde (Schardong & Custódio, 2022, S. 6). Der Verifier kann dann eigenständig entscheiden, ob er dem Issuer vertraut oder nicht (Schardong & Custódio, 2022, S. 6). Revokationsinformationen sind öffentlich verfügbar und können anonym abgefragt werden, ohne dass die spezifische Identität des Abfragenden offengelegt wird (Schardong & Custódio, 2022, S. 6).

Ein kritischer technischer Enabler ist der Zero Knowledge Proof (ZKP), ein kryptographisches Verfahren, welches es einem Prover ermöglicht, einem Verifier zu beweisen, dass eine Aussage wahr ist, ohne die zugrundeliegenden Daten preiszugeben (Schardong & Custódio, 2022, S. 6). Dies ermöglicht es Holdern beispielsweise, nachzuweisen, dass sie älter als 18 Jahre sind, ohne ihr genaues Geburtsdatum preiszugeben (Schardong & Custódio, 2022, S. 6).

2.2 Blockchain-Technologie

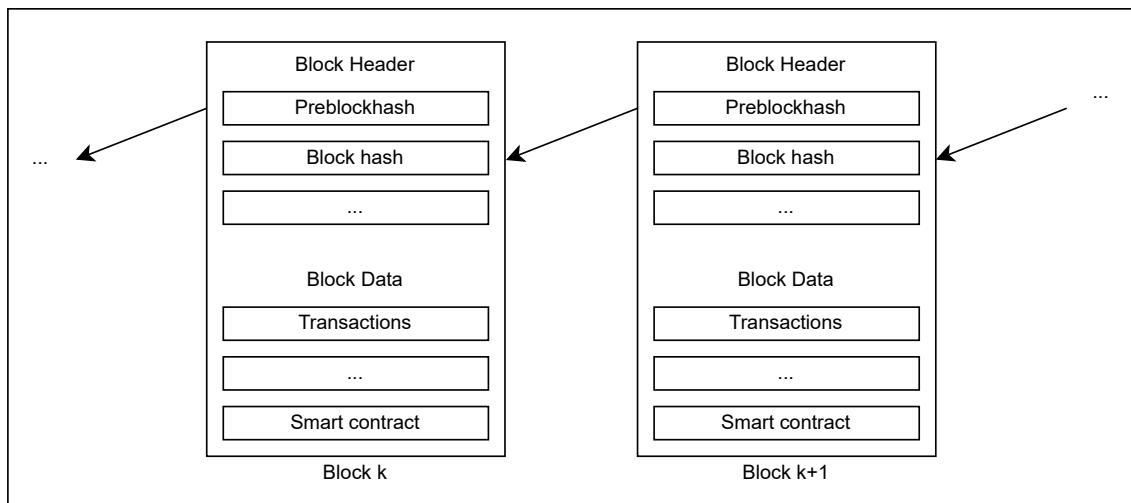
Die Blockchain-Technologie wird in der Literatur als eine spezifische Ausprägung der DLT klassifiziert, die die für die SSI-Architektur erforderlichen Charakteristika Zuverlässigkeit, Unveränderlichkeit, Transparenz, Irreversibilität und Auditierbarkeit durch zeitchronologische Anordnung erfüllt (Strüker et al., 2021, S. 27).

Strukturell besteht eine Blockchain aus einer sequenziellen Kette von Datenblöcken. Jeder Block setzt sich aus den beiden Hauptkomponenten Block-Header und Block-Body zusammen (Moosavi et al., 2024, S. 647).

Der Block-Header enthält essenzielle Metadaten, darunter den Zeitstempel, eineNonce(fürProof-of-Work-Systeme)undvorallemdenkryptographischenHash-WertdesvorherigenBlocks(PreviousBlockHash)(Moosavietal.,2024,S.648).Durchdiese kryptographische Verkettung (Abbildung 4) entsteht eine untrennbare Historie. Die Modifikation eines einzigen Transaktionsdatensatzes würde den Hash des Blocks verändern, was wiederum die Ungültigkeit aller nachfolgenden Blöcke zur Folge hätte (Moosavi et al., 2024, S. 648). Der Merkle-Root-Hash im Block-Header ermöglicht somit eine effiziente Verifikation der Datenintegrität, ohne alle Knoten der Merkle-Tree-Struktur überprüfen zu müssen (Kuznetsov et al., 2024, S. 4).

Der Block-Body enthält die eigentlichen Transaktionen, die häufig über einen Merkle Tree (Hash-Baum) effizient organisiert sind (Moosavi et al., 2024, S. 648).

Abbildung 4
Blockchain Sequenz



Anmerkung. Eigene Darstellung nach Moosavi et al. (2024, S. 648).

2.2.1 Klassifizierung von Ledger-Architekturen

Blockchain-Netzwerke werden primär nach dem Zugriffsprivileg in die beiden Modelle Permissionless und Permissioned klassifiziert (Abbildung 5) (Moosavi et al., 2024, S. 649).

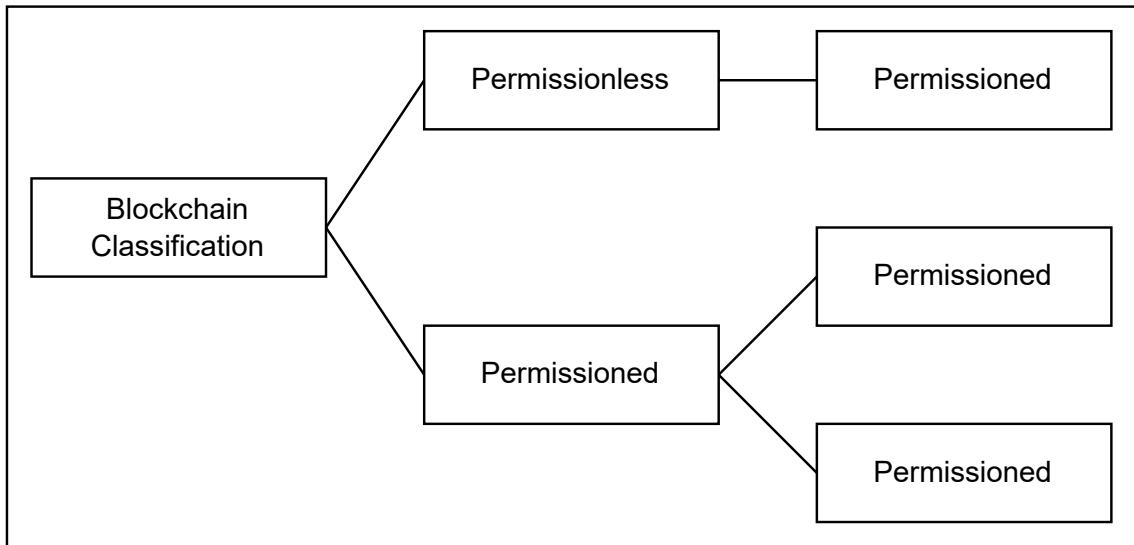
Permissionless Blockchains sind vollständig offen, jeder Teilnehmer kann dem Netzwerk beitreten, lesen und schreiben (Moosavi et al., 2024, S. 649).

Permissioned Blockchains werden in zwei Hauptgruppen unterteilt, private Blockchains und Konsortium-Blockchains. Bei der ersten Gruppe hat nur eine ausgewählte Anzahl von

Knoten Zugriff auf das Netzwerk und kann Transaktionen validieren. Diese Einschränkung beschleunigt die Transaktionen, sodass selbst bei wachsender Blockchain die Transaktionszeit effizient bleibt. In dieser Art von Blockchain können vollständig sichere Transaktionen durchgeführt werden. Der Hauptnachteil einer privaten Blockchain besteht darin, dass die Validierung zentralisiert wird, da sie unter der Kontrolle einer einzelnen Organisation steht. Eine Konsortium-Blockchain weist einige Besonderheiten einer öffentlichen oder privaten Blockchain auf, bei der mehrere Organisationen das Netzwerk kontrollieren (Moosavi et al., 2024, S. 649–650).

Permissioned Blockchains werden für Identitätsmanagement-Lösungen oft präferiert, da sie Datenschutzanforderungen, wie Data Segregation, besser abbilden können (Naghmouchi & Laurent, 2025, S. 8–10).

Abbildung 5
Blockchain Klassifizierung



Anmerkung. Eigene Darstellung nach Moosavi et al. (2024, S. 649).

2.2.2 Die Rolle der Blockchain in SSI-Systemen

In einer SSI-Architektur fungiert die Blockchain nicht als Speicherort für Identitätsdaten selbst (Strüker et al., 2021, S. 27), sondern als VDR (Satybaldy, Ferdous & Nowostawski, 2024, S. 7). Sie dient als manipulationssichere Quelle für kryptographisches Material, das für die Verifikation von Credentials notwendig ist, ohne dabei personenbezogene Daten preiszugeben (Strüker et al., 2021, S. 27).

Der Ledger übernimmt im SSI-Kontext spezifische, klar definierte Aufgaben:

DPKI: Anstatt sich auf eine zentrale Certificate Authority (CA) zu verlassen, werden Public Keys (in Form von DID-Dokumenten) auf der Blockchain verankert (Strüker et al., 2021, S. 17). Da der Ledger manipulationssicher ist, kann jeder Verifier den Public Key eines Issuers zweifelsfrei abrufen und dessen Signaturen validieren, ohne den Issuer direkt kontaktieren zu müssen (Strüker et al., 2021, S. 21).

Schema- und Credential-Definitionen: Um Interoperabilität zu gewährleisten, werden die Strukturdaten von Credentials (Schemata) auf dem Ledger publiziert. Dies stellt sicher, dass alle Parteien dieselbe Datenstruktur für ein spezifisches Credential verwenden und interpretieren können (Strüker et al., 2021, S. 19).

Revocation Registries: Ein kritischer Aspekt ist die Revocation von Credentials. SSI-Systeme nutzen hierfür oft kryptographische Akkumulatoren auf der Blockchain (Strüker et al., 2021, S. 26–28). Diese erlauben es einem Holder, mittels eines Non-Revocation Proofs zu beweisen, dass sein Credential nicht widerrufen wurde, ohne dass der Issuer bei jeder Prüfung kontaktiert werden muss (Strüker et al., 2021, S. 29).

2.3 Post-Quantum Kryptografie

PQC befasst sich mit kryptographischen Verfahren, die entwickelt wurden, um Angriffen durch Quantencomputer zu widerstehen (Mamatha, Dimri & Sinha, 2024, S. 2), da davon ausgegangen wird, dass populäre Public-Key-Algorithmen wie RSA, DSA und ECDSA durch den Shor-Algorithmus auf hinreichend leistungsfähigen Quantencomputern gebrochen werden (Bernstein et al., 2009, S. 1–2). Während praktische Quantencomputer noch in der Entwicklung sind (Chhetri et al., 2025, S. 1–2), arbeiten Regierungen, Normungsorganisationen und Akteure der Industrie bereits an Übergangsstrategien, um sensible Informationen und KRITIS langfristig zu schützen (Mamatha, Dimri & Sinha, 2024, S. 6).

2.3.1 Abgrenzung zu Quantum Cryptography

PQC und Quantum Cryptography unterscheiden sich grundlegend in Scope, Sicherheitsgrundlagen und praktischer Anwendbarkeit. Während Quantum Cryptography ausschließlich der Schlüsselexpansion, der Erweiterung eines kurzen gemeinsamen Geheimnisses in einen längeren Bitstrom, dient, deckt PQC ein breites Spektrum kryptographischer Aufgaben ab, einschließlich digitaler Signaturen und Public-Key-Verschlüsselung (Bernstein et al., 2009, S. 13–14).

Ein entscheidender Unterschied liegt in den Sicherheitsannahmen. Während PQC teilweise auf Vermutungen beruht, basiert Quantum Cryptography theoretisch auf etablierten Gesetzen der Quantenmechanik. Praktisch jedoch erfordert Quantum Cryptography spezialisierte, kostspielige Hardware und direkte Glasfaserverbindungen, wohingegen PQC-Systeme mit konventioneller Computertechnik implementierbar sind und über bestehende Netzwerke funktionieren. Diese Unterschiede machen PQC zur praktikablen Lösung für den Schutz vor Quantencomputern im Internet-Maßstab, während Quantum Cryptography auf spezialisierten Anwendungen mit hohem Budgetaufwand beschränkt bleibt (Bernstein et al., 2009, S. 13–14).

2.3.2 Kategorien von PQC-Algorithmen

Die prominentesten PQC-Algorithmen lassen sich in mehrere Kategorien einteilen, die in Tabelle 1 dargestellt sind.

Tabelle 1
Übersicht ausgewählter PQC-Algorithmus-Kategorien

Typ	Vorteil	Nachteil	Algorithmus
Gitter	Hohe Betriebsgeschwindigkeit	Schwierige Parametereinstellungen	CRYSTALS-DILITHIUM, FALCON
Code	Kleine Signaturgrößen, Hohe Betriebsgeschwindigkeit	Große Schlüsselgrößen	Classic McEliece, BIKE, HQC

Fortsetzung auf nächster Seite

Tabelle 1 (Fortsetzung)

Typ	Vorteil	Nachteil	Algorithmus
Hash	Sicherheitsnachweis möglich	Große Signaturgrößen	SPHINCS+, PICNIC
Multivariat	Schnelle Ver- und Entschlüsselungsgeschwindigkeiten	Große Schlüsselgrößen	Rainbow

Anmerkung. Eigene Darstellung nach Mamatha, Dimri und Sinha (2024, S. 2).

Die gitterbasierte Kryptografie nutzt die Schwierigkeit mathematischer Probleme auf Gittern wie dem Shortest Vector Problem oder dem Learning With Errors Problem, die als quantenresistent gelten (Mamatha, Dimri & Sinha, 2024, S. 2). Bei der codebasierten Kryptografie werden fehlerkorrigierende Codes wie Goppa-Codes oder McEliece-Kryptosysteme eingesetzt, deren Sicherheit auf der Schwierigkeit des Dekodieren bestimmter linearer Codes beruht (Mamatha, Dimri & Sinha, 2024, S. 2). Die hashbasierte Kryptografie basiert auf kryptographischen Hash-Funktionen zur Bereitstellung digitaler Signaturen und ist nicht direkt durch Quantenangriffe gefährdet (Mamatha, Dimri & Sinha, 2024, S. 2). Die multivariate Polynom-Kryptografie hingegen stützt sich auf die Schwierigkeit, Systeme multivariater Polynomgleichungen zu lösen, was als quantenresistent gilt.

2.3.3 NIST-Standardisierung der PQC-Algorithmen

Im August 2024 hat das NIST drei Federal Information Processing Standards veröffentlicht, die quantenresistente kryptographische Verfahren für die Bundesbehörden der USA verpflichtend einführen (National Institute of Standards and Technology, 2024a, 2024b, 2024c, S. i). Mit FIPS 203 wurde ML-KEM als gitterbasiertes Schlüsselkapselungsmechanismus standardisiert (National Institute of Standards and Technology, 2024b, S. i). FIPS 204 spezifiziert ML-DSA als gitterbasierten digitalen Signaturalgorithmus, der aus CRYSTALS-DILITHIUM abgeleitet wurde (National Institute of Standards and Technology, 2024a, S. 1) und als sicher gegen Angreifer mit großen Quantencomputern gilt (National Institute of Standards and Technology, 2024a, S. i). Als drittes Verfahren wurde mit FIPS 205 der zustandslose hashbasierte Signaturalgorithmus SLH-DSA standardisiert, der auf SPHINCS+ basiert und dessen Sicherheit auf der Schwierigkeit des Findens von Urbildern für kryptographische Hashfunktionen beruht (National Institute of Standards and Technology, 2024c, S. 1).

2.4 Kritische Infrastrukturen und Compliance

Gemäß Bundesministerium der Justiz (2009, § 2 Abs. 10 Satz 1) werden KRITIS als Einrichtungen, Anlagen oder Teile davon definiert, die zwei zentrale Kriterien erfüllen. Zum einen müssen sie einem der Sektoren Energie, Informationstechnik und Telekommunikation, Transport und Verkehr, Gesundheit, Wasser, Ernährung, Finanz- und Versicherungswesen oder Siedlungsabfallsorgung angehören. Zum anderen ist eine hohe Bedeutung für das Funktionieren des Gemeinwesens erforderlich, da ein Ausfall oder eine Beeinträchtigung dieser Systeme erhebliche Versorgungsengpässe oder Gefährdungen für die öffentliche Sicherheit zur Folge hätte.

Die konkrete Bestimmung, welche Einrichtungen und Anlagen als KRITIS gelten, wird nach Bundesministerium der Justiz (2009, § 10 Abs. 1) durch eine Rechtsverordnung des Bundesministeriums des Innern, für Bau und Heimat festgelegt. Diese Verordnung erfolgt nach Anhörung von Vertretern der Wissenschaft, der betroffenen Betreiber und der betroffenen Wirtschaftsverbände im Einvernehmen mit zahlreichen weiteren Bundesministerien. Die Verordnung bestimmt dabei die in den jeweiligen Sektoren als kritisch anzusehenden Dienstleistungen sowie deren als bedeutend anzusehende Versorgungsgrade. Zur Operationalisierung dieser Kriterien werden branchenspezifische Schwellenwerte herangezogen, die für jede als kritisch anzusehende Dienstleistung im jeweiligen Sektor individuell festgelegt werden. Diese systematische Klassifizierung ermöglicht es, die Schutzmaßnahmen und Compliance-Anforderungen angemessen auf die Risiken und Bedeutsamkeit der einzelnen Infrastrukturen abzustimmen.

2.4.1 Regulatorische Anforderungen

Die regulatorischen Anforderungen an die Informationssicherheit von KRITIS erfordern den Einsatz kryptographischer Verfahren, die dem Stand der Technik entsprechen. Gemäß Bundesministerium der Justiz (2009, § 8a Abs. 1) muss ein KRITIS-Betreiber die zur Erbringung der kritischen Dienstleistung notwendigen informationstechnischen Systeme und Komponenten so gestalten, dass diese gegen Störungen der Verfügbarkeit, Integrität, Authentizität und Vertraulichkeit angemessen geschützt sind (BSI, 2024, S. 7). Mit Blick auf die fortschreitende Entwicklung von Quantencomputern gewinnt die Transition zu PQC zunehmend an Bedeutung für die Erfüllung regulatorischer Anforderungen.

Die DSGVO fordert in Artikel 32 geeignete technische und organisatorische Maßnahmen zum Schutz personenbezogener Daten, einschließlich Verschlüsselung als Sicherheitsmaßnahme nach dem Stand der Technik (Das europäische Parlament und der Rat der

europäischen Union, 2016, S. 51). Diese Anforderung wird insbesondere für langfristig schutzbedürftige Daten relevant, da diese angesichts der „Harvest Now, Decrypt Later“-Strategie (Geremew & Mohammad, 2024, S. 344–345) gefährdet sind. Zur Begegnung dieser Bedrohung wird eine rechtzeitige Migration zu quantensicheren Signaturverfahren empfohlen, insbesondere für Anwendungsfälle mit langen Migrationszeiten (BSI, 2025b, S. 21–22).

Für die Implementierung kryptographischer Systeme in KRITIS-Umgebungen ist dabei die Kryptoagilität (Kapitel 2.5) von zentraler Bedeutung, die es ermöglicht, kryptographische Algorithmen, Schlüssellängen und Schlüsselgenerierungsverfahren ohne wesentliche Änderungen am Gesamtsystem auszutauschen (BSI, 2025b, S. xii). Das BSI empfiehlt gemeinsam mit europäischen Partnerbehörden, die Migration zu quantensicheren Verfahren für hochsensitive Anwendungen bis spätestens Ende 2030 abzuschließen (BSI, 2025b, S. 30).

2.4.2 Compliance-Vorgaben

Das BSI konkretisiert die regulatorischen Anforderungen durch die Veröffentlichung technischer Richtlinien, welche die Grundlage für die Erfüllung der Compliance-Vorgaben bilden (BSI, 2025b, S. 18–21). Die Technische Richtlinie TR-02102-1 empfiehlt konkrete kryptographische Verfahren und Schlüssellängen, die das erforderliche Sicherheitsniveau von mindestens 120 Bit erreichen (BSI, 2025b, S. 18–19).

Im Kontext der kryptographischen Sicherheit fordert ISO/IEC (2022, Control A.8.24), dass Regeln für den wirksamen Einsatz von Kryptografie, einschließlich Schlüsselverwaltung, definiert und umgesetzt werden müssen (ISO/IEC, 2022, S. 17). Dies umfasst auch die Berücksichtigung zukünftiger kryptographischer Entwicklungen und der Notwendigkeit zur Transition.

Die vorliegende Technische Richtlinie empfiehlt den hybriden Einsatz quantensicherer Schlüsseleinigungsverfahren in Kombination mit klassischen Verfahren zum langfristigen Schutz der Vertraulichkeit (BSI, 2025b, S. 21).

Die konkrete Umsetzung dieser Anforderungen erfordert dabei die Berücksichtigung spezifischer Empfehlungen für Transportprotokolle. Standards für die Nutzung quantensicherer Verfahren in Transport Layer Security (TLS) werden derzeit entwickelt und erprobt, wobei das BSI beabsichtigt, quantensichere Verfahren in hybrider Nutzung mit empfohlenen klassischen Verfahren zu empfehlen, sobald geeignete Standards verabschiedet wurden (BSI, 2025a, S. 6).

2.5 Kryptoagilität

Kryptoagilität beschreibt die Fähigkeit kryptographischer Systeme, verwendete Verfahren, Parameter und Implementierungen flexibel und mit vertretbarem Aufwand an neue Sicherheitsanforderungen anzupassen, ohne die grundlegende Systemarchitektur vollständig überarbeiten zu müssen (Mehrez & El Omri, 2018, S. 101–102). Diese Fähigkeit ist zentral, um auf neu entdeckte Schwachstellen, technologische Fortschritte oder disruptive Bedrohungen wie Quantencomputer zu reagieren. Dies erfolgt durch den Austausch kryptographischer Algorithmen oder die Ergänzung zusätzlicher Schutzmechanismen (Mehrez & El Omri, 2018, S. 99–100; Marchesi, Marchesi & Tonelli, 2025, S. 1).

Um die in Tabelle 2 dargestellten Eigenschaften der Kryptoagilität zu erfüllen, basieren Kryptoagile Architekturen auf modularen Komponenten (Mehrez & El Omri, 2018, S. 102–103).

Tabelle 2
Eigenschaften der Kryptoagilität

Eigenschaft	Beschreibung
Extensibility	Fähigkeit, neue Algorithmen oder Parameter effizient zum System hinzuzufügen
Removability	Fähigkeit, veraltete oder anfällige kryptographische Verfahren kontrolliert zu deaktivieren
Fungibility	Leichte Austauschbarkeit von Sicherheitskomponenten durch standardisierte Identifikatoren
Interoperability	Kompatibilität zwischen verschiedenen Implementierungen basierend auf Spezifikationen
Updateability	Sichere Aktualisierung oder Korrektur kryptographischer Algorithmen Implementierungen
Flexibility	Grad der Anpassungsfähigkeit je nach Implementierungsebene (Hardware < Betriebssystem < Anwendung < Skriptsprachen)
Compatibility	Neue Softwaremodule und Patches müssen auf bestehender Hardware lauffähig sein

Fortsetzung auf nächster Seite

Tabelle 2 (Fortsetzung)

Eigenschaft	Beschreibung
Reversibility	Fähigkeit, fehlgeschlagene Softwareupdates rückgängig zu machen und zu vorherigen, funktionierenden Versionen zurückzukehren
Transition Mechanisms	Geschützte Aushandlung neuer Algorithmen gegen Downgrade-Attacken
Backward Compatibility	Unterstützung älterer Algorithmen während begrenzter Übergangsphasen

Anmerkung. Eigene Darstellung basierend auf Mehrez und El Omri (2018, S. 102–103).

Kreutzer, Wolf und Waidner (2024, S. 667) heben hervor, dass kryptoagile Systeme eine Voraussetzung für langfristige Sicherheit und für die Aufnahme kryptographischer Innovationen darstellen. Entsprechende Sicherheitsarchitekturen müssen daher bereits im Design vorsehen, dass kryptographische Komponenten aktualisiert, kombiniert und betrieblich migriert werden können, um auf Innovationsdruck, Erosion bestehender Verfahren und inkompatible Standards reagieren zu können (Kreutzer, Wolf & Waidner, 2024, S. 667–671). Kryptoagilität wird in diesem Sinne als Sichtweisen verschiedener Handlungsfelder verstanden, das sich von kryptographischen Bibliotheken über Systemarchitekturen bis hin zu Entscheidungsprozessen und Governance-Strukturen erstreckt (Kreutzer, Wolf & Waidner, 2024, S. 669–670).

3 Methodik

3.1 Systematische Literaturrecherche

Die systematische Literaturrecherche dieser Masterarbeit folgt ausgewählten Methoden der PRISMA 2020 Richtlinien zur strukturierten Identifikation, Selektion, Bewertung und Synthese einschlägiger Studien (Page et al., 2021, S. 1). Dieses Vorgehen bietet einen strukturierten Ansatz zur Durchführung und Dokumentation von Literaturrecherchen, welcher die Qualität und Vollständigkeit der Berichterstattung verbessert und einen transparenten und reproduzierbaren Prozess gewährleistet (Page et al., 2021, S. 1), wodurch eine belastbare Grundlage für die Analyse der Forschungslücke (Kapitel 1.2) und die Ableitung der Forschungsfragen (Kapitel 1.3) dieser Masterarbeit geschaffen wird.

Die systematische Literaturrecherche wurde in zwei zeitlich getrennten Iterationen durchgeführt, um den dynamischen Charakter des Forschungsfeldes zu adressieren und die Aktualität der Wissensbasis sicherzustellen. Beide Iterationen sind in Anhang 2 ausführlich dokumentiert und dienen ausschließlich der Problemidentifikation, der Ergründung des Forschungsstands sowie der Ableitung der Forschungslücke und initialen Forschungsfragen.

Der konsolidierte Suchprozess in der Datenbank EBSCO führt über beide Phasen hinweg zur Identifikation von insgesamt 95 potenziellen Quellen. Davon entfallen 61 Publikationen auf die erste Iteration im Mai 2025 (siehe Abbildung A-2) und 34 Publikationen auf die zweite Iteration im November 2025 (siehe Abbildung A-4). Die methodische Konsistenz wird dabei durch die Anwendung identischer Suchstrategien und Selektionskriterien in beiden Zeiträumen sichergestellt.

Nach der Bereinigung von Duplikaten und der Anwendung der Ein- und Ausschlusskriterien verbleibt ein fokussierter Bestand an hochrelevanten Studien. Eine detaillierte Übersicht der identifizierten Quellen findet sich in Tabelle A-7 für die erste und Tabelle A-10 für die zweite Phase. Die vollständige Aufschlüsselung der Selektionsschritte sowie die zugehörigen Flussdiagramme sind zudem gesammelt in Anhang 2 aufgeführt.

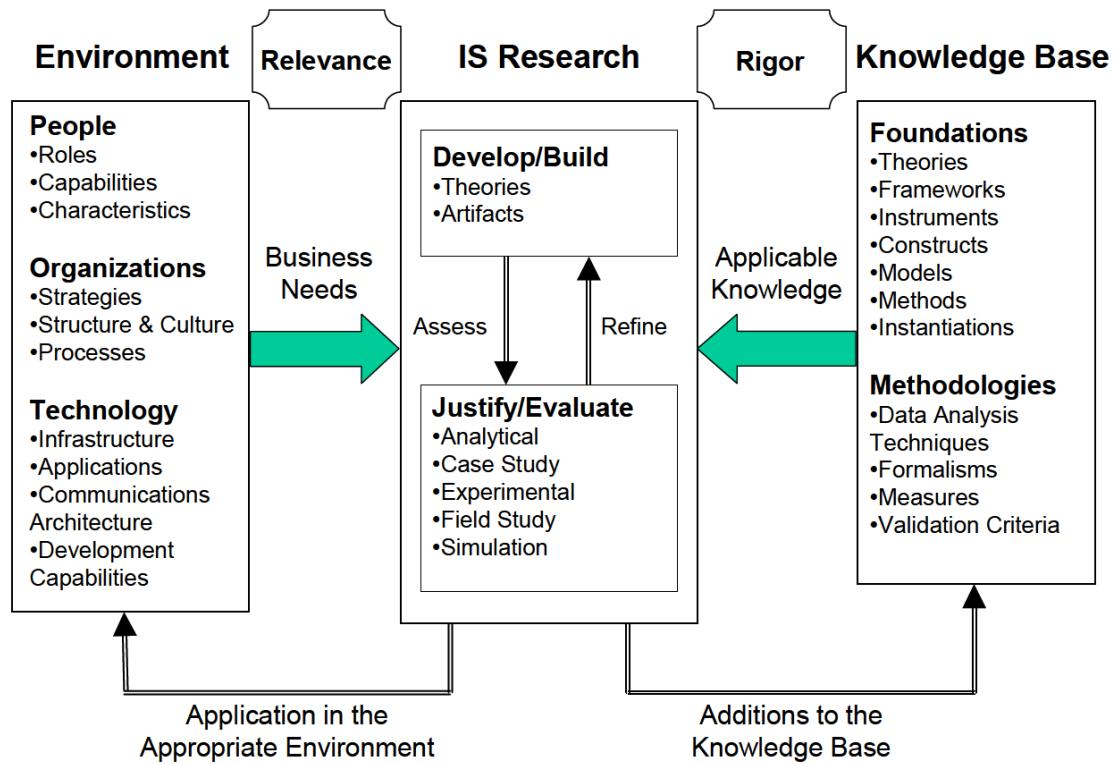
3.2 Design Science Research

DSR bildet den methodischen Rahmen dieser Masterarbeit und stellt neben der verhaltenswissenschaftlichen Forschung ein eigenständiges Forschungsparadigma dar (Hevner et al., 2004, S. 75). Während verhaltenswissenschaftliche Ansätze Theorien zur Erklärung

oder Vorhersage entwickeln, fokussiert DSR auf die Erschaffung innovativer Artefakte zur Erweiterung menschlicher und organisatorischer Fähigkeiten (Hevner et al., 2004, S. 75). Fundamental ist DSR ein lösungsorientiertes Paradigma (Hevner et al., 2004, S. 76), dessen Prinzip darin besteht, Wissen durch den Bau und die Anwendung eines Artefakts zu gewinnen (Hevner et al., 2004, S. 82).

Das Information Systems Research Framework (Abbildung 6) nach Hevner et al. (2004, S. 80) strukturiert den Forschungsprozess durch drei Hauptkomponenten.

Abbildung 6
Information Systems Research Framework



Anmerkung. Aus Hevner et al. (2004, S. 80).

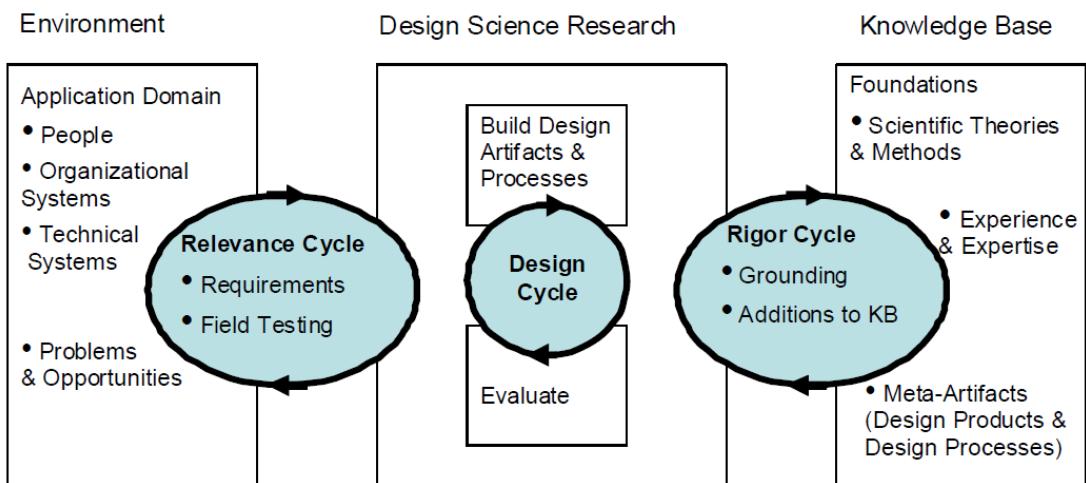
Die Environment-Komponente definiert den Problemraum (Simon, 1996, S. 108) mit Menschen, Organisationen und Technologien sowie den Geschäftsanforderungen (Silver, Markus & Beath, 1995, S. 7–11). Für diese Arbeit bilden KRITIS das Environment mit ihren Anforderungen an Post-Quantum-Sicherheit und selbstbestimmte Identitätsverwaltung. Die IS Research-Komponente umfasst Build- und Evaluate-Aktivitäten für Artefakte (Hevner et al., 2004, S. 80). Hier erfolgt die Instanziierung und Evaluation eines PQC-fähigen SSI-Prototypen, der speziell für den Einsatz in KRITIS konzipiert ist. Die Knowledge Base liefert

Foundations (Theorien, Frameworks, Konstrukte) und Methodologies (Evaluationsmethoden) (Hevner et al., 2004, S. 80). Sie umfasst für diese Arbeit SSI-Standards, NIST-PQC-Algorithmen und KRITIS-Anforderungen. Rigor wird durch angemessene Anwendung dieser Wissensbasis erreicht (Hevner et al., 2004, S. 80).

3.2.1 Zyklen

Hevner (2007, S. 88) identifiziert drei eng verbundene Aktivitätszyklen, die in jedem DSR-Projekt präsent sein müssen (Abbildung 7).

Abbildung 7
Design Science Research Zyklen



Anmerkung. Aus Hevner (2007, S. 88).

Der Relevance Cycle initiiert DSR mit Anforderungen aus der Anwendungsdomäne und fordert Field Testing des Outputs (Hevner, 2007, S. 88–89). In dieser Arbeit manifestiert sich dies durch die iterative Identifikation von KRITIS-Anforderungen an Sicherheit, Compliance und Kryptoagilität in zwei Implementierungszyklen. Dazu gehören quantenresistente Kommunikations- und Signaturverfahren, die Einhaltung relevanter Regulierungs-vorgaben sowie die Wahrung von Datenschutz und technischer Resilienz. Die entwickelten Artefakte werden kontinuierlich in Laborumgebungen getestet, wobei Erkenntnisse aus der ersten Iteration die Designziele der zweiten Iteration prägen. Der Rigor Cycle verbindet DSR-Aktivitäten mit der Wissensbasis, um Innovation zu gewährleisten (Hevner, 2007, S.

88–90). In dieser Arbeit operationalisiert dieser Zyklus die kontinuierliche Integration etablierter SSI-Frameworks, NIST-standardisierter PQC-Algorithmen und KRITIS-Standards durch wissenschaftlich fundierte Software-Engineering-Patterns, die „Defense in Depth“ (Alsaqour et al., 2021, S. 242–243) etablieren. Der Design Cycle strukturiert die Artefaktentwicklung als iterativen Prozess zwischen Konstruktion und Evaluation (Hevner, 2007, S. 90–91). In dieser Arbeit erfolgt dies in zwei aufeinanderfolgenden Iterationen, wobei Evaluationsergebnisse als Designziele der Folge-Iteration operationalisiert werden.

3.2.2 Artefakte

March und Smith (1995, S. 255) identifizieren vier Artefakttypen: Constructs, Models, Methods und Instantiations. Constructs stellen die Sprache bereit, in der Probleme und Lösungen definiert werden, und beeinflussen die Problemkonzeption (Hevner et al., 2004, S. 78, 83). Models repräsentieren das Designproblem und den Lösungsraum, unterstützen das Problemverständnis und ermöglichen die Erkundung von Designentscheidungen (Hevner et al., 2004, S. 78–79). Methods definieren Prozesse zur Problemlösung, von formalen Algorithmen bis zu Best-Practice-Beschreibungen (Hevner et al., 2004, S. 79). Instantiations demonstrieren Machbarkeit durch Implementierung in einem funktionierenden System und liefern den Beweis durch Konstruktion (Hevner et al., 2004, S. 79, 84).

Diese Arbeit entwickelt eine Instantiation in Form eines funktionsfähigen Prototypen eines blockchain-basierten SSI-Systems mit integrierter PQC für KRITIS, welcher die technische Machbarkeit und praktische Anwendbarkeit des Ansatzes demonstriert.

3.2.3 Richtlinien

Hevner et al. (2004, S. 82) formulieren sieben Richtlinien für DSR, die jeweils in einer qualitätsvollen Forschungsarbeit adressiert werden sollten. Diese Richtlinien bilden einen strukturierten Rahmen zur Durchführung und Bewertung von Design-Science-Forschung und gewährleisten, dass sowohl wissenschaftliche Rigor als auch praktische Relevanz erreicht werden.

Guideline 1: Design as an Artifact - Design-Science-Forschung muss ein brauchbares Artefakt produzieren (Hevner et al., 2004, S. 82). Diese Arbeit erfüllt dies durch eine funktionsfähige Instantiation eines blockchain-basierten SSI-Systems mit integrierter PQC.

Guideline 2: Problem Relevance - Design-Science-Forschung zielt auf technologiebasierte Lösungen für relevante Geschäftsprobleme ab (Hevner et al., 2004, S. 84–85). Die Relevanz ergibt sich aus der Quantenbedrohung für KRITIS-Kryptografie und dem Bedarf an datenschutzfreundlichen Identitätslösungen.

Guideline 3: Design Evaluation - Die Nützlichkeit und Wirksamkeit eines Design-Artefakts müssen rigoros demonstriert werden (Hevner et al., 2004, S. 85). Diese Arbeit validiert Funktionalität, Compliance und Kryptoagilität.

Guideline 4: Research Contributions - DSR muss klare Beiträge zu Design-Artefakt, Foundations oder Methodologies liefern (Hevner et al., 2004, S. 87). Der Beitrag liegt in der neuartigen Integration von PQC in SSI-Systemen für KRITIS-Kontexte.

Guideline 5: Research Rigor - DSR beruht auf rigoroser Anwendung von Methoden in Konstruktion und Evaluation (Hevner et al., 2004, S. 87). Diese Arbeit nutzt für die Konstruktion etablierte SSI-Frameworks, NIST-standardisierte PQC-Algorithmen, wissenschaftlich fundierte Software-Engineering-Patterns und das FEDS-Framework für die Evaluation.

Guideline 6: Design as a Search Process - Die Suche nach einem effektiven Artefakt erfordert iterative Exploration unter Berücksichtigung der Problemumgebung (Hevner et al., 2004, S. 87–88). Der Design Cycle dieser Arbeit manifestiert sich als Prozess mit zwei Iterationen, bei dem Erkenntnisse aus der ersten Iteration die Designziele der nachfolgenden Iteration prägen.

Guideline 7: Communication of Research - Design-Science-Forschung muss effektiv sowohl für technologie-orientierte als auch für management-orientierte Audiences präsentiert werden (Hevner et al., 2004, S. 90). Diese Arbeit adressiert dies durch eine umfassende Dokumentation des entwickelten Prototypen, systematische Darstellung der Evaluationsergebnisse sowie die explizite Ableitung praktischer Implikationen für KRITIS.

3.3 FEDS-Framework

Nach Venable, Pries-Heje und Baskerville (2016, S. 77) ist die Evaluation von Design-Artefakten eine Schlüsselaktivität des DSR, ohne die die Forschung auf der Ebene theoretischer Annahmen über die Utility eines Artefakts verbleibt, ohne Evidenz für dessen tatsächliche Funktionsfähigkeit zu liefern. Um diese Lücke zu schließen und Rigor sicherzustellen, folgt diese Arbeit dem FEDS-Framework nach Venable, Pries-Heje und Baskerville (2016, S. 77–84), welches den Evaluationsprozess in vier iterative Schritte unterteilt, um die Strategie passgenau auf die spezifischen Projektrisiken abzustimmen.

Das FEDS-Framework unterscheidet dabei fundamental zwischen zwei Dimensionen der Evaluation: der funktionalen Absicht und dem Paradigma (Venable, Pries-Heje & Baskerville, 2016, S. S. 77–84). Hinsichtlich der Absicht wird zwischen formativer Evaluation, die der kontinuierlichen Verbesserung während der Entwicklung dient, und summativer Evaluation, die eine abschließende Bewertung der Zielerreichung vornimmt, differenziert (Venable, Pries-Heje & Baskerville, 2016, S. S. 77–84). Bezuglich des Paradigmas unterscheidet das Framework zwischen Artificial Evaluation, die in kontrollierten Laborumgebungen stattfindet, und Naturalistic Evaluation, welche die Artefakte in realen organisatorischen Umfeldern unter echten Bedingungen prüft (Venable, Pries-Heje & Baskerville, 2016, S. S. 77–84). Im Folgenden wird der vierstufige FEDS-Prozess für die vorliegende Arbeit erläutert.

3.3.1 Explikation der Evaluationsziele

Der erste Schritt des Frameworks verlangt die Explikation der Evaluationsziele, um Konflikte zwischen konkurrierenden Anforderungen wie Rigor, Risikominimierung und Effizienz aufzulösen (Venable, Pries-Heje & Baskerville, 2016, S. 82–83). Für die Entwicklung eines blockchain-basierten SSI-Prototypen mit PQC im KRITIS-Kontext ergeben sich hieraus spezifische Prioritäten.

Rigour (Efficacy vs. Effectiveness): Das primäre Ziel dieser Arbeit ist der Nachweis der „Technical Feasibility“ durch rigorose Demonstration der „Efficacy“. Dies bedeutet den rigorosen Beleg, dass das instanzierte Artefakt (die PQC-Integration in SSI) den beobachteten Effekt (sichere, quantenresistente Identitätsverifikation) kausal verursacht und nicht externe Störfaktoren (Venable, Pries-Heje & Baskerville, 2016, S. 82). Da es sich bei KRITIS-Komponenten um sicherheitskritische Infrastruktur handelt, muss vor einem Feldtest („Effectiveness“ in realer Umgebung) zwingend die funktionale Korrektheit in einer kontrollierten Umgebung bewiesen werden, um „False Positives“, eine fälschliche Annahme der Sicherheit, auszuschließen (Venable, Pries-Heje & Baskerville, 2016, S. 79).

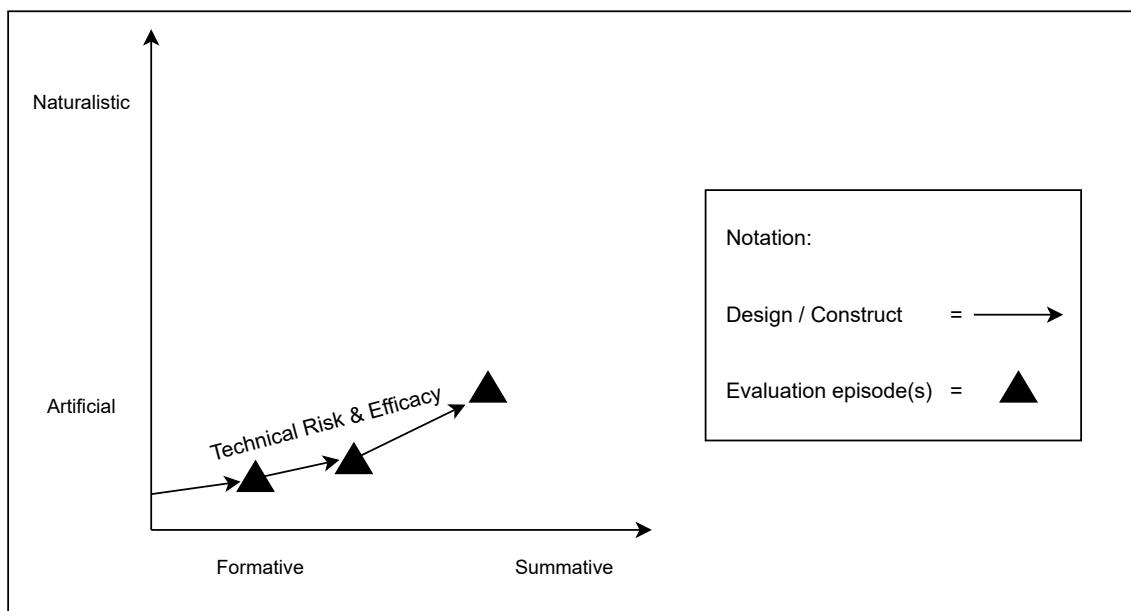
Risikoreduktion (Technical Risk): Das Hauptrisiko dieser Arbeit ist technischer Natur. Es besteht die Unsicherheit, ob die PQC-Algorithmen (ML-DSA, ML-KEM) technisch in bestehende SSI-Frameworks (Hyperledger Aries) integriert werden können, ohne deren Kernfunktionen zu brechen. Soziale Risiken (z.B. Benutzerakzeptanz der Wallet-App) werden als nachrangig eingestuft und nicht evaluiert.

Ausschluss von Effizienz-Zielen: In Anlehnung an die Design-Science-Leitlinien wird explizit darauf hingewiesen, dass eine quantitative Performance-Evaluation („Efficiency“) nicht Ziel dieser Arbeit ist. Die verwendeten PQC-Referenzimplementierungen befinden sich in einem frühen Stadium, weshalb Laufzeitmessungen keine valide Aussagekraft für zukünftige produktive Systeme hätten (Demir, Bilgin & Onbasli, 2025, S. 3–5).

3.3.2 Wahl der Evaluationsstrategie

Basierend auf den identifizierten Zielen und Risiken wird für diese Arbeit die Technical Risk & Efficacy Strategy (Venable, Pries-Heje & Baskerville, 2016, S. 81) gewählt, welche mit formativen, artifiziellen Tests beginnt und in einer summativen, artifiziellen Evaluation endet (Abbildung 8).

Abbildung 8
Gewählte Evaluationsstrategie im FEDS-Framework



Anmerkung. Eigene Darstellung in Anlehnung an Venable, Pries-Heje & Baskerville (2016, S. 80).

Diese Strategie ist nach Venable, Pries-Heje und Baskerville (2016, S. 82) dann indiziert, wenn das primäre Entwicklungsrisiko technischer Natur ist und Evaluationen in realen Umgebungen aus Sicherheits- oder Kostengründen nicht durchführbar sind. Ein naturalistischer Ansatz („Human Risk & Effectiveness“) wird verworfen, da der Zugriff auf reale KRITIS-Netzwerke für experimentelle kryptografische Prototypen ethisch und regulatorisch nicht vertretbar ist und die Technologie noch nicht den Reifegrad für Endanwender-tests besitzt.

3.3.3 Bestimmung der zu evaluierenden Eigenschaften

Der dritte Schritt definiert die konkreten Eigenschaften, die evaluiert werden sollen (Venable, Pries-Heje & Baskerville, 2016, S. 83–84). Für das entwickelte Artefakt leiten sich diese direkt aus den in Kapitel 4.2 definierten Anforderungen ab:

1. **Fidelity (Funktionale Korrektheit):** Das System muss in der Lage sein, den vollständigen SSI-Lebenszyklus (Issuance, Verification, Revocation) unter Verwendung von PQC-Signaturen fehlerfrei durchzuführen. Es wird geprüft, ob die Artefakte (Agenten, Ledger, Wallet) spezifikationsgemäß interagieren.
2. **KRITIS-Compliance (Sicherheit):** Es wird evaluiert, ob die implementierten Mechanismen den regulatorischen Vorgaben entsprechen. Dies umfasst die strikte Durchsetzung von TLS 1.3, die Verwendung hybrider Verfahren und die Netzwerksegmentierung.
3. **Interoperabilität:** Die Fähigkeit des modifizierten Systems, Standard-DIDComm-Nachrichten trotz der veränderten kryptografischen Payload zu verarbeiten, ist ein kritisches Kriterium für die Efficacy.

3.3.4 Design der individuellen Evaluationsepisoden

Der vierte Schritt umfasst das Design konkreter Evaluationsepisoden (Venable, Pries-Heje & Baskerville, 2016, S. 84). Für diese Arbeit wurden drei diskrete Episoden definiert, die parallel zu den Entwicklungszyklen verlaufen und in Tabelle 3 dem jeweiligen Arbeitsfortschritt zugeordnet sind.

Tabelle 3*Evaluationsepisoden nach dem FEDS-Framework*

Episode	Phase & Art	Fokus / Methode
1	Erste Iteration (Kapitel 4.6) <i>Formativ, Artifiziell</i>	Log-Analyse, Zertifikatsvalidierung, TLS-Handshake-Analyse und Infrastruktur-Initialisierung in Form von White-Box Tests nach Myers, Sandler und Badgett (2012, S. 10).
2	Zweite Iteration (Kapitel 5.4) <i>Formativ, Artifiziell</i>	Integrationstests der Plugin-Architektur, DIDComm-Verarbeitung und Validierung der PQC-Signaturen in Form von White-Box Tests nach Myers, Sandler und Badgett (2012, S. 10).
3	Finales Artefakt (Kapitel 6) <i>Summativ, Artifiziell</i>	Requirement Tracing und Validierung der KRITIS-Compliance am integrierten Gesamtsystem.

Anmerkung. Eigene Darstellung der Evaluationsstrategie in Anlehnung an das FEDS-Framework von Venable, Pries-Heje und Baskerville (2016).

Episode 1 (Formativ): Diese Episode erfolgt parallel zur ersten Iteration und fokussiert sich auf die Validierung der TLS. Da in dieser Phase fundamentale Infrastrukturkomponenten wie Sidecar-Proxies entwickelt werden, dient die formative Evaluation primär dazu, Designfehler so früh wie möglich zu identifizieren (Venable, Pries-Heje & Baskerville, 2016, S. 82–83). Methodisch erfolgt dies durch die Analyse von Handshake-Protokollen und Cipher-Suites.

Episode 2 (Formativ): In der zweiten Iteration verlagert sich der Fokus auf den Application-Layer. Hier wird formativ geprüft, ob die entwickelten Python-Plugins korrekt in den Aries Cloud Agent Python (ACA-Py)-Core geladen werden und ob die erweiterte Kryptografiebibliothek liboqs korrekt angesprochen wird.

Episode 3 (Summativ): Die abschließende Evaluation in Kapitel 6 führt alle Komponenten zusammen. Sie dient dem rigorosen Nachweis, dass das Gesamtsystem die eingangs definierten Forschungsfragen beantwortet. Hierbei wird geprüft, ob die Efficacy des PQC-SSI-Prototypen für KRITIS-Anwendungsfälle gegeben ist, ohne reale Risiken einzugehen.

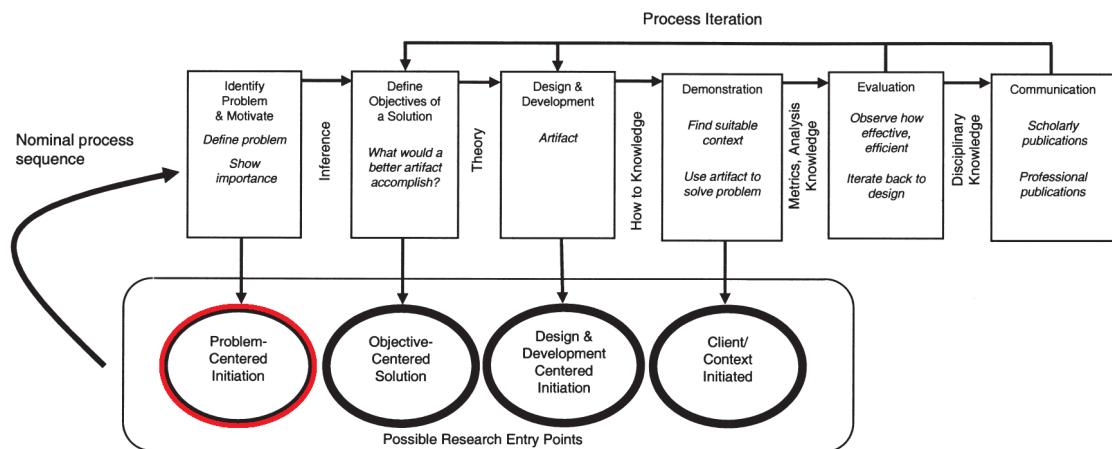
3.4 DSRM Prozessmodell

Zur Sicherstellung einer rigorosen methodischen Fundierung orientiert sich der Forschungsablauf dieser Arbeit am DSRM Prozessmodell nach Peffers et al. (2007, S. 46), welches einen Rahmen für die Durchführung und Präsentation von DSR bereitstellt um die wissenschaftliche Validität von Design-Artefakten zu gewährleisten.

Da die vorliegende Arbeit durch die spezifischen Bedrohungen des Quantencomputings für bestehende Infrastrukturen und die daraus resultierenden regulatorischen Anforderungen für KRITIS motiviert ist (Kapitel 1.1), folgt das Forschungsvorhaben einer „Problem-Centered Initiation“. Dies entspricht dem klassischen Einstieg in den DSRM-Prozess über die erste Phase (Abbildung 9) (Peffers et al., 2007, S. 56).

Neben der „Objective-Centered Solution“, die durch einen Bedarf in der Industrie oder Forschung ausgelöst wird (Phase 2), definieren die Peffers et al. (2007, S. 56) eine „Design & Development Centered Initiation“ auf Basis existierender Artefakte für explizite Problembereiche (Phase 3), sowie eine „Client-/Context-Initiated Solution“, welche auf der Beobachtung einer praktischen Lösung basiert (Phase 4), als drei weitere Einstiegspunkte.

Abbildung 9
DSRM Process Model



Anmerkung. Adaptiert aus Peffers et al. (2007, S. 54).

Die Operationalisierung der sechs Phasen des DSRM beginnt mit der Phase „Problem Identification and Motivation“ (Peffers et al., 2007, S. 52–55). Die Definition des spezifischen Forschungsproblems und dessen Relevanz für KRITIS wurde hierfür in Kapitel 1.1 dargelegt.

Darauf aufbauend werden in der Phase „Define the Objectives for a Solution“ (Peffers et al., 2007, S. 55) aus der Problemanalyse konkrete Forschungsfragen (Kapitel 1.3) und designrelevante Ziele für jede Iteration (Kapitel 4.1 und Kapitel 5.1) abgeleitet, welche als Grundlage für die späteren Evaluationsphasen dienen.

Den Kern der Arbeit bildet die Phase „Design and Development“ (Peffers et al., 2007, S. 55), welche die Konzeption der Architektur sowie die technische Implementierung des PQC-SSI-Prototypen in beiden Iterationen umfasst (Kapitel 4 und 5). Diese Phase operationalisiert die systematische Umsetzung der in den vorherigen Phasen definierten Anforderungen in ein funktionsfähiges Artefakt.

Die Eignung dieses Artefakts zur Problemlösung wird im Rahmen der Phase „Demonstration“ (Peffers et al., 2007, S. 55) validiert. Diese erfolgt durch formative Evaluationsepisoden während der beiden Iterationen, welche schrittweise die Funktionsfähigkeit des Artefakts durch modulare Tests unter kontrollierten Bedingungen (Kapitel 4.6 und 5.4), bis hin zum vollständigen Prototypen (Kapitel 6) nachweisen.

Daran anschließend wird in der Phase „Evaluation“ (Peffers et al., 2007, S. 56) strukturiert nach dem FEDS-Framework (Kapitel 3.3) in mehreren diskrete Evaluationsepisoden eine systematische Bewertung des Artefakts vorgenommen. Formative Episoden während beider Iterationen (Kapitel 4.6 und 5.4) adressieren technische Einzelfunktionen und Designfehler frühzeitig, die abschließende summative Evaluation in Kapitel 6 validiert systematisch die funktionale Korrektheit, KRITIS-Compliance und Kryptoagilität des integrierten Gesamtsystems.

Den Abschluss bildet die Phase „Communication“ (Peffers et al., 2007, S. 56), in der die Ergebnisse, der Designprozess und die Evaluation durch die vorliegende schriftliche Ausarbeitung dokumentiert und der wissenschaftlichen Gemeinschaft zur Verfügung gestellt werden.

Obwohl das Modell sequenziell dargestellt ist, handelt es sich bei der Entwicklung in den Kapiteln 4 und 5 um einen iterativen Prozess, der Rücksprünge von der Evaluation zur Design-Phase erlaubt, um das Artefakt schrittweise zu verfeinern (Peffers et al., 2007, S. 56).

4 Erste Iteration der Artefaktentwicklung

4.1 Designziele dieser Iteration

Die Designziele dieser Iteration leiten sich unmittelbar aus den in Kapitel 1.3 definierten Forschungsfragen ab und werden operationalisiert durch eine schichtenbasierte Architektur, die die Quantensicherheit auf der Transportebene verankert.

Im Fokus von FF1 (Systemarchitektur & Compliance) steht die Etablierung einer modularen Architektur mit klarer Separation zwischen den SSI-Agenten, der DLT-Infrastruktur und der quantensicheren Transportschicht auf Basis ausgewählter Frameworks und Technologien. Das Design soll die Kernherausforderung, PQC nicht-invasiv und modular zu integrieren, adressieren.

Bezüglich FF2 (Algorithmenauswahl und Sicherheitsbewertung) liegt das Designziel auf der Erprobung quantenresistenter kryptografischer Primitive für die Transportebene auf Basis standardisierter Algorithmen und hybrider Schlüsselaustauschverfahren. Die formative Evaluierung soll dabei die technische Machbarkeit dieser Algorithmen in der Infrastruktur validieren und Erkenntnisse für die nachgelagerte Iteration generieren.

Für FF3 (Kryptografische Agilität) zielt diese Iteration auf die architektonische Vorbereitung der Austauschbarkeit kryptografischer Algorithmen ab. Das Design soll durch modulare Infrastrukturkomponenten die Voraussetzungen für zukünftige Algorithmenupdates ohne grundlegende Systemumgestaltung schaffen.

4.2 Anforderungsanalyse

4.2.1 Funktionale Anforderungen

Basierend auf der Analyse von Nokhbeh Zaeem et al. (2021, S. 130) lassen sich für das SSI-System sechs zentrale funktionale Anforderungen identifizieren, die den vollständigen Lebenszyklus digitaler Identitätsnachweise abdecken (Tabelle 4).

Tabelle 4
Funktionale Anforderungen an SSI-Systeme

Nr.	Funktionale Anforderung	Beschreibung
FR1	Issuer Discovery	Das System muss die Auffindbarkeit von publizierten Credential Schemata des Issuers digitaler Identitätsnachweise ermöglichen.
FR2	Connection Creation	Das System muss Verbindungen zwischen den Akteuren des SSI-Ökosystems etablieren können.
FR3	Credential Creation	Das System muss Funktionalität zur Erstellung und Ausstellung digitaler Credentials bereitstellen.
FR4	Verification with Credentials	Das System muss einen Verifikationsprozess zwischen Identity Holder, Verifier und Blockchain-basierter VDR durch Validierung eines Identitätsnachweises ermöglichen.
FR5	Credential Revocation	Das System muss die Funktionalität zum Widerruf von Credentials unterstützen.
FR6	Credential Deletion	Das System muss die Funktionalität zur Löschung von Credentials unterstützen.

Anmerkung. Eigene Darstellung auf Basis der Auflistungen und des Sequenzdiagramms in Anlehnung an Nokhbeh Zaeem et al. (2021, S. 130–132).

4.2.2 KRITIS-spezifische Compliance-Anforderungen

Die in Tabelle 5 konsolidierten Anforderungen definieren den normativen Rahmen für die Gestaltung und Evaluierung des PQC-SSI-Prototypen im Kontext KRITIS.

Im Bereich der kryptografischen Verfahren (Nr. CR1–CR4) basieren die Vorgaben primär auf den Technischen Richtlinien des BSI. Für die Migration auf PQC ist insbesondere die Wahl spezifischer Parameter-Sets für ML-DSA (NIST Level 3/5) und ML-KEM (Level 3/5) sowie die zwingende Implementierung hybrider Schlüsseleinigung vorgeschrieben, um sowohl Integrität als auch langfristige Vertraulichkeit gegen Quantencomputer-Angriffe zu gewährleisten (BSI, 2025b, Kap. 2.2, 2.4, 5.3.4.2). Ergänzend fordert die TR-02102-2 den Einsatz moderner Transportverschlüsselung via TLS 1.3, um durch Perfect Forward Secrecy die Kommunikationskanäle abzusichern (BSI, 2025a, Kap. 3.2).

Hinsichtlich der Betriebssicherheit (Nr. CR5–CR6) leiten sich die Anforderungen direkt aus nationalen Gesetzen und internationalen Standards ab. Essenziell für KRITIS-Betreiber ist hierbei die Implementierung effektiver SzA durch umfassende Protokollierung sicherheitsrelevanter Ereignisse gemäß § 8a BSIG (BSI, 2024, Nr. 101, 103). Flankierend schreibt die ISO/IEC 27001 eine strikte logische Netzsegmentierung vor, um die Ausbreitung potenzieller Sicherheitsvorfälle innerhalb der Infrastruktur zu begrenzen (ISO/IEC, 2022, Control A.8.22).

Die dritte Säule bildet der Datenschutz (Nr. CR7–CR9) auf Basis der DSGVO. Hierbei stehen Prinzipien wie „Privacy by Design“ gemäß Art. 25 und Datenminimierung nach Art. 5 im Fokus. Zudem muss das Recht auf Löschung nach Art. 17 durch geeignete Architekturmuster, etwa die Trennung von Identifikatoren und Inhaltsdaten, technisch gewährleistet werden (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 5, 17, 25).

Tabelle 5

Compliance Anforderungen an SSI-Systeme im KRITIS-Kontext

Nr.	Compliance Anforderung	Beschreibung
CR1	Einhaltung spezifischer Parameter-Sets für ML-DSA	Zur Gewährleistung der vom BSI geforderten Sicherheitsniveaus dürfen für das Verfahren ML-DSA ausschließlich die Parameter-Sets verwendet werden, die den NIST Security Strength Categories 3 oder 5 entsprechen. Konkret sind dies ML-DSA-65 oder ML-DSA-87 (BSI, 2025b, Kap. 5.3.4.2)
CR2	Einhaltung spezifischer Parameter-Sets für ML-KEM	Für den langfristigen Schutz vertraulicher Informationen mittels des gitterbasierten Schlüsselkapselungsverfahrens ML-KEM dürfen gemäß BSI-Einschätzung ausschließlich Parametersätze verwendet werden, die den NIST Security Strength Categories 3 oder 5 entsprechen. Zulässig sind demnach ML-KEM-768 sowie ML-KEM-1024 (BSI, 2025b, Kap. 2.4.3).

Fortsetzung auf nächster Seite

Tabelle 5 (Fortsetzung)

Nr.	Compliance Anforderung	Beschreibung
CR3	Implementierung hybrider Schlüsselseinigung	Um langfristige Vertraulichkeit und den Schutz vor der „Harvest Now, Decrypt Later“-Strategie (Geremew & Mohammad, 2024, S. 344–345) zu gewährleisten, muss für die Schlüsselseinigung zwingend ein hybrides Verfahren implementiert werden, das ein anerkanntes klassisches Verfahren mit einem empfohlenen PQC-KEM kombiniert (BSI, 2025b, Kap. 2.2, 2.4).
CR4	Bevorzugte Verwendung von TLS 1.3	Für die Absicherung der Transportebene wird gemäß BSI (2025a, Kap. 3.2) vorrangig das Protokoll TLS 1.3 empfohlen, da es PFS erzwingt und auf unsichere Cipher-Suites verzichtet.
CR5	Protokollierung sicherheitsrelevanter Ereignisse	Sicherheitsrelevante Ereignisse müssen auf System- und Netzebene zentral protokolliert werden, um eine zeitnahe Erkennung von Angriffen zu ermöglichen (BSI, 2024, Nr. 101, 103).
CR6	Logische Netzsegmentierung	Gruppen von Informationsdiensten, Benutzern und Informationssystemen sollten in den Netzwerken der Organisation getrennt werden (ISO/IEC, 2022, Control A.8.22).
CR7	Datenschutz durch Technikgestaltung (Privacy by Design)	Gemäß Das europäische Parlament und der Rat der europäischen Union (2016, Art. 25) sind bereits bei der Entwicklung des Systems geeignete technische Maßnahmen zu treffen, die die Datenschutzgrundsätze addressieren.
CR8	Grundsatz der Datenminimierung	Personenbezogene Daten müssen dem Zweck angemessen und auf das notwendige Maß beschränkt sein. (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 5).

Fortsetzung auf nächster Seite

Tabelle 5 (Fortsetzung)

Nr.	Compliance Anforderung	Beschreibung
CR9	Recht auf Löschung	Die betroffene Person hat das Recht, von dem Verantwortlichen die unverzügliche Löschung sie betreffender personenbezogener Daten zu verlangen. Der Verantwortliche ist verpflichtet, personenbezogene Daten unverzüglich zu löschen (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 17).

Anmerkung. Eigene Darstellung basierend auf BSI (2024, 2025a, 2025b), Das europäische Parlament und der Rat der europäischen Union (2016) und ISO/IEC (2022).

4.3 Framework- und Technologie-Auswahl

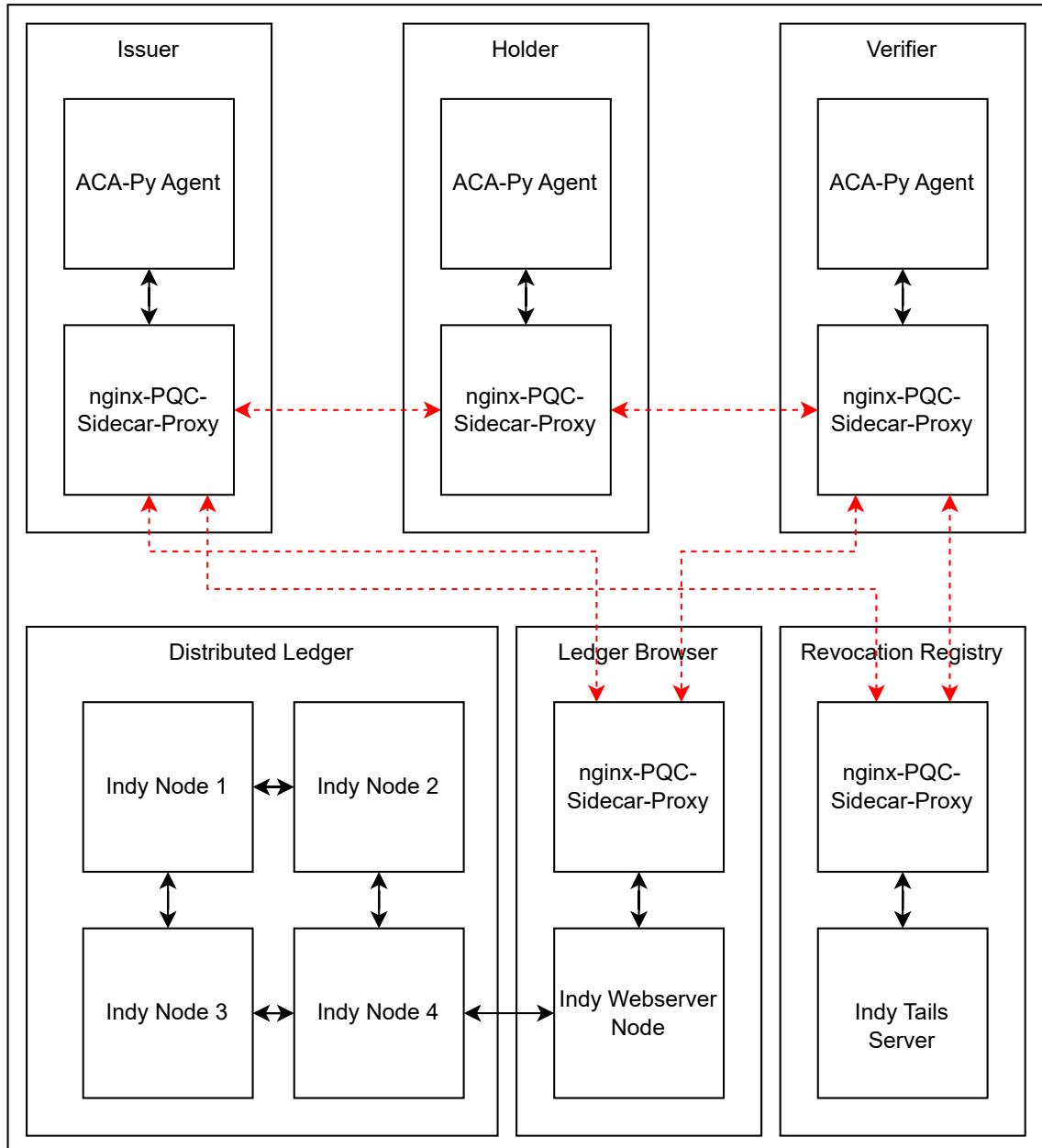
Wie von Ghosh et al. (2021, S. 3) demonstriert, setzt auch die vorliegende Arbeit auf existierende Konzepte und Tools für dezentrale Identitätsverwaltung als Bausteine für die Entwicklung des Prototypen. Hierbei anzumerken ist, dass die Entwicklung von DID und VC sich noch in einem frühen Stadium befindet, wobei nur relativ wenige Implementierungen verfügbar sind (Mazzocca et al., 2025, S. 9). Die systematische Auswahl der Frameworks und Technologien, bestehend aus der DLT-Plattform, dem SSI-Framework, der Kryptografiebibliothek, der Revocation-Infrastruktur und dem Sidecar-Proxy, wird transparent in Anhang 3.1 dokumentiert und begründet.

4.4 Architekturentwurf

4.4.1 Gesamtarchitektur

Der Architekturentwurf dieser Iteration operationalisiert die in Kapitel 4.1 definierten Designziele durch eine dreischichtige, containerbasierte Systemarchitektur, die PQC auf der Transportebene verankert. Abbildung 10 visualisiert die Zielarchitektur, bestehend aus der DLT-Schicht mit vier Hyperledger-Indy-Validator-Nodes samt Ledger Browser, der Revocation-Schicht mit dediziertem Tails-Server sowie der SSI-Agenten-Schicht mit drei ACA-Py-Instanzen in den Rollen Issuer, Holder und Verifier.

Abbildung 10
Gesamtarchitekturentwurf der ersten Iteration



Anmerkung. Eigene Darstellung.

Das zentrale Architekturprinzip bildet das Sidecar-Proxy Pattern (Anhang 3.1.5). Jede extern erreichbare Komponente wird durch einen nginx-PQC-Sidecar-Proxy geschützt, der TLS-1.3-Verbindungen mit hybrider Schlüsseleinigung terminiert und im Fallback ECC (X25519MLKEM768:x25519) unterstützt. Die Authentifizierung erfolgt über ML-DSA-65-

signierte X.509-Zertifikate. In Abbildung 10 wird diese quantensichere Kommunikation durch die rot-gestrichelten Verbindungsfeile zwischen den Sidecar-Proxies visualisiert.

Die Backend-Dienste (Indy-Nodes, Webserver, Tails-Server und ACA-Py-Agents) verbleiben innerhalb isolierter interner Docker-Netzwerke und kommunizieren ausschließlich über unverschlüsseltes HTTP. Externe Zugriffe erfolgen ausschließlich über das gemeinsame Netzwerk „sidecar_proxy“, das als quantensichere Kommunikationsdomäne fungiert und eine strikte Netzsegmentierung gemäß CR6 (Tabelle 5) gewährleistet.

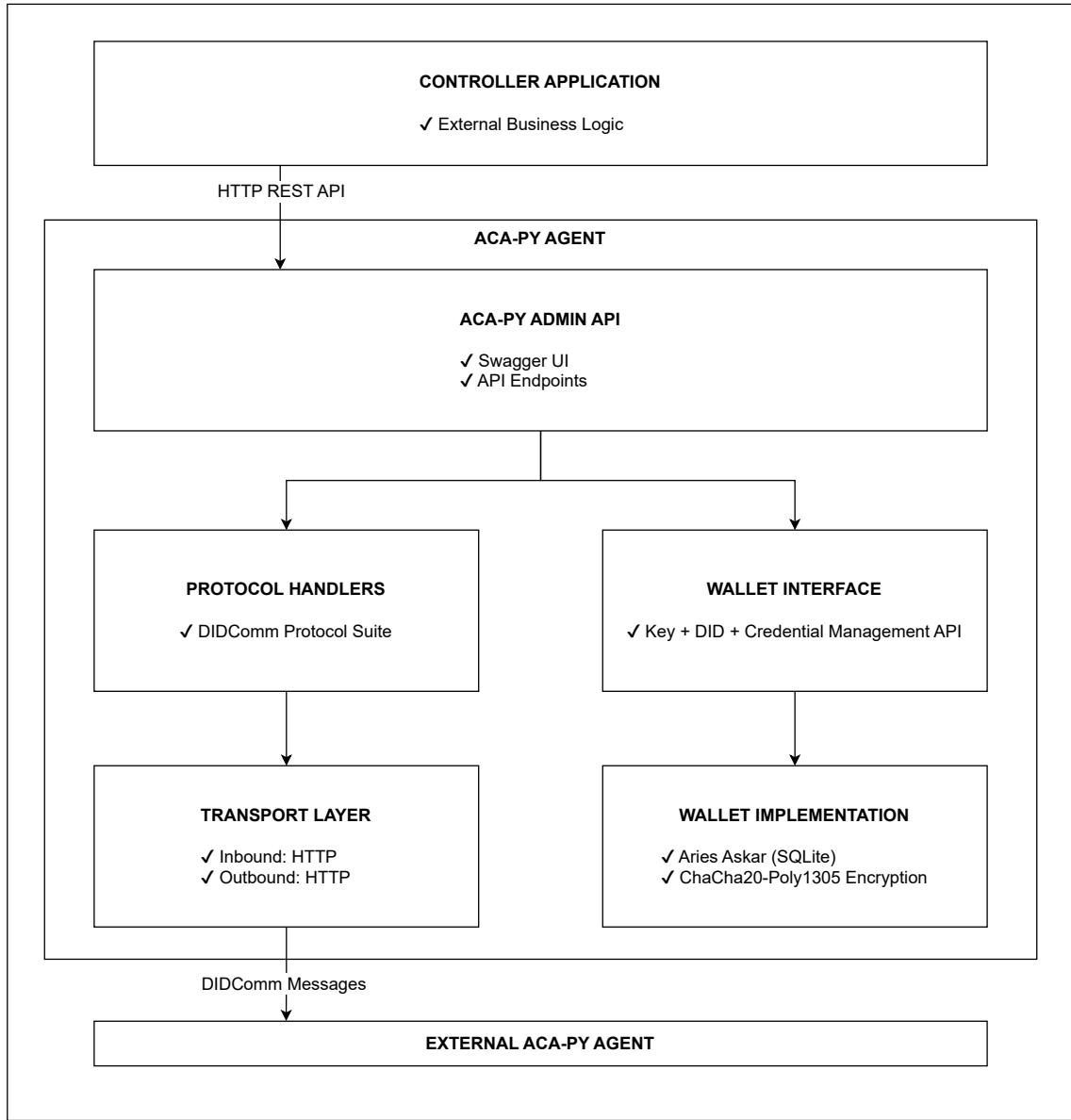
4.4.2 ACA-Py Applikationsarchitektur

Die Architektur von ACA-Py folgt einem modularen Designansatz, der durch eine strikte Trennung zwischen der generischen Protokollebene und der anwendungsspezifischen Geschäftslogik gekennzeichnet ist. Dieses Architekturmuster entkoppelt die kryptografischen Kernfunktionen und das DIDComm-Messaging (Agent) von der Steuerungslogik (Controller) (openwallet-foundation, n. d. m). Abbildung 11 visualisiert diese High-Level-Applikationsarchitektur und verdeutlicht die Interaktion der Schichten.

Layer 1 bildet eine HTTP-REST-API (Admin API), die über das „server.py“-Modul implementiert wird (openwallet-foundation, n. d. a). Sie ermöglicht es Controller-Anwendungen den Agent über standardisierte Endpoints wie „/out-of-band/create-invitation“ (openwallet-foundation, n. d. h) oder „/issue-credential-2.0/send“ (openwallet-foundation, n. d. f) zu steuern, ohne direkt mit der Python-Codebasis zu interagieren. Die Endpunkte werden hierbei durch mehrere „routes.py“-Module umgesetzt.

Layer 2 implementiert die SSI-Geschäftslogik als Protocol Handler durch mehrere „manager.py“-Module, die Aries Request for Comments (RFC)s umsetzen. Das Out-of-Band Protocol (RFC 0434) (decentralized-identity, n. d. b) generiert Invitation-Nachrichten mit „did peer 4-DID“ (openwallet-foundation, n. d. g), das DID Exchange Protocol (RFC 0023) (decentralized-identity, n. d. a) authentifiziert Agent-Verbindungen mittels ED25519-Signaturen (openwallet-foundation, n. d. d), und das Issue Credential Protocol (RFC 0453) (decentralized-identity, n. d. c) erstellt AnonCreds-Credentials (openwallet-foundation, n. d. e).

Abbildung 11
ACA-Py High Level Applikationsarchitektur



Anmerkung. Eigene Darstellung basierend auf decentralized-identity (n. d. a, n. d. b, n. d. c) und openwallet-foundation (n. d. a, n. d. b, n. d. c, n. d. f, n. d. g, n. d. h, n. d. i, n. d. j, n. d. k, n. d. l, n. d. m).

Layer 3 abstrahiert die Schlüsselverwaltung durch das Aries-Askar-Wallet (openwallet-foundation, n. d. k), das ED25519-Schlüsselpaare für Signaturen und X25519-Schlüsselpaare für Key Agreement generiert (openwallet-foundation, n. d. l), diese mit Multicodec-Präfixen kodiert (openwallet-foundation, n. d. l) und verschlüsselt in einer SQLite-Datenbank ablegt (openwallet-foundation, n. d. c), wobei ChaCha20-Poly1305-Authenticated-Encryption verwendet wird (openwallet-foundation, n. d. b).

Layer 4 realisiert das DIDComm-Messaging über die Methoden „pack_message()“ und „unpack_message()“ (openwallet-foundation, n. d. b). Die Daten werden anschließend mittels HTTP-basierter Transport-Mechanismen für Inbound- (openwallet-foundation, n. d. i) und Outbound-Kommunikation (openwallet-foundation, n. d. j) übertragen, wobei in der klassischen Architektur kein quantensicheres TLS verwendet wird.

4.5 Implementierung

Die Implementierung realisiert die Zertifikatsinfrastruktur mit ML-DSA-Signaturen, PQC-Sidecar-Proxies mit TLS 1.3 und hybrider Schlüsseleinigung, die Hyperledger-Indy-DLT-Schicht, die Revocation Registry sowie drei ACA-Py-Agenten, die mittels Docker-Compose mit expliziter Netzsegmentierung orchestriert werden. Die hierfür genutzte Entwicklungsumgebung ist in Anhang 3.2.1 dokumentiert.

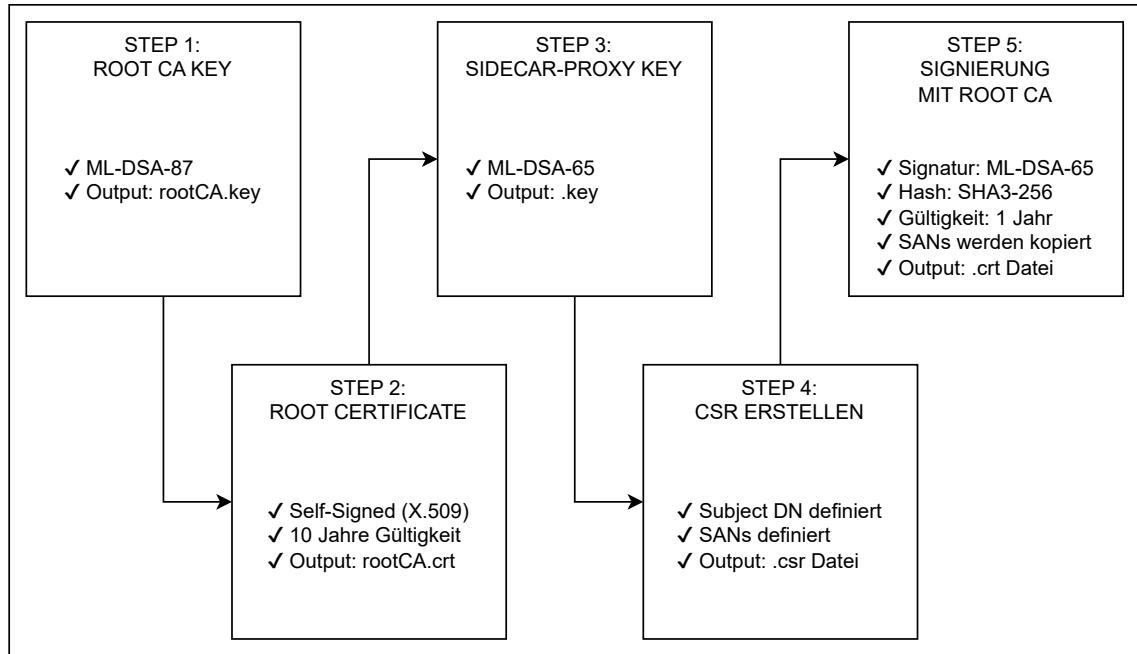
4.5.1 Zertifikatsstruktur

Die Zertifikatsinfrastruktur für die PQC-Sidecar-Proxies basiert auf einer selbstsignierten Root CA, die mit dem Post-Quantum-Signaturalgorithmus ML-DSA-87 erstellt wurde. Sie dient als Trust Anchor für alle in der Architektur verwendeten TLS-Zertifikate und gewährleistet, dass sämtliche Zertifikatssignaturen quantenresistent nach der höchsten NIST Sicherheitsstufe 5 sind (National Institute of Standards and Technology, 2024a, S. 15).

Das Zertifikaterstellungsverfahren folgt einem fünfschrittigen Workflow, wie in Abbildung 12 dargestellt.

Abbildung 12

Zertifikatserstellungsworkflow für PQC-basierte Sidecar-Proxies

*Anmerkung.* Eigene Darstellung.

Zunächst wird der Root CA-Schlüssel generiert (Step 1), gefolgt von der Erstellung des selbstsignierten Root CA-Zertifikats mit einer Gültigkeit von zehn Jahren (Step 2). Anschließend werden für jeden Sidecar-Proxy dedizierte Schlüssel mit ML-DSA-65 generiert (Step 3), die ein besseres Verhältnis zwischen Sicherheit und Speicherbedarf bieten (National Institute of Standards and Technology, 2024a, S. 16). Für jeden Sidecar-Proxy wird ein Certificate Signing Request erstellt, der die erforderlichen Subject Alternative Names enthält (Step 4), bevor die Zertifikate abschließend von der Root CA mit ML-DSA-65 und SHA3-256 signiert werden (Step 5). Die detaillierte Implementierung dieses Workflows ist in Anhang 3.2.2 dokumentiert.

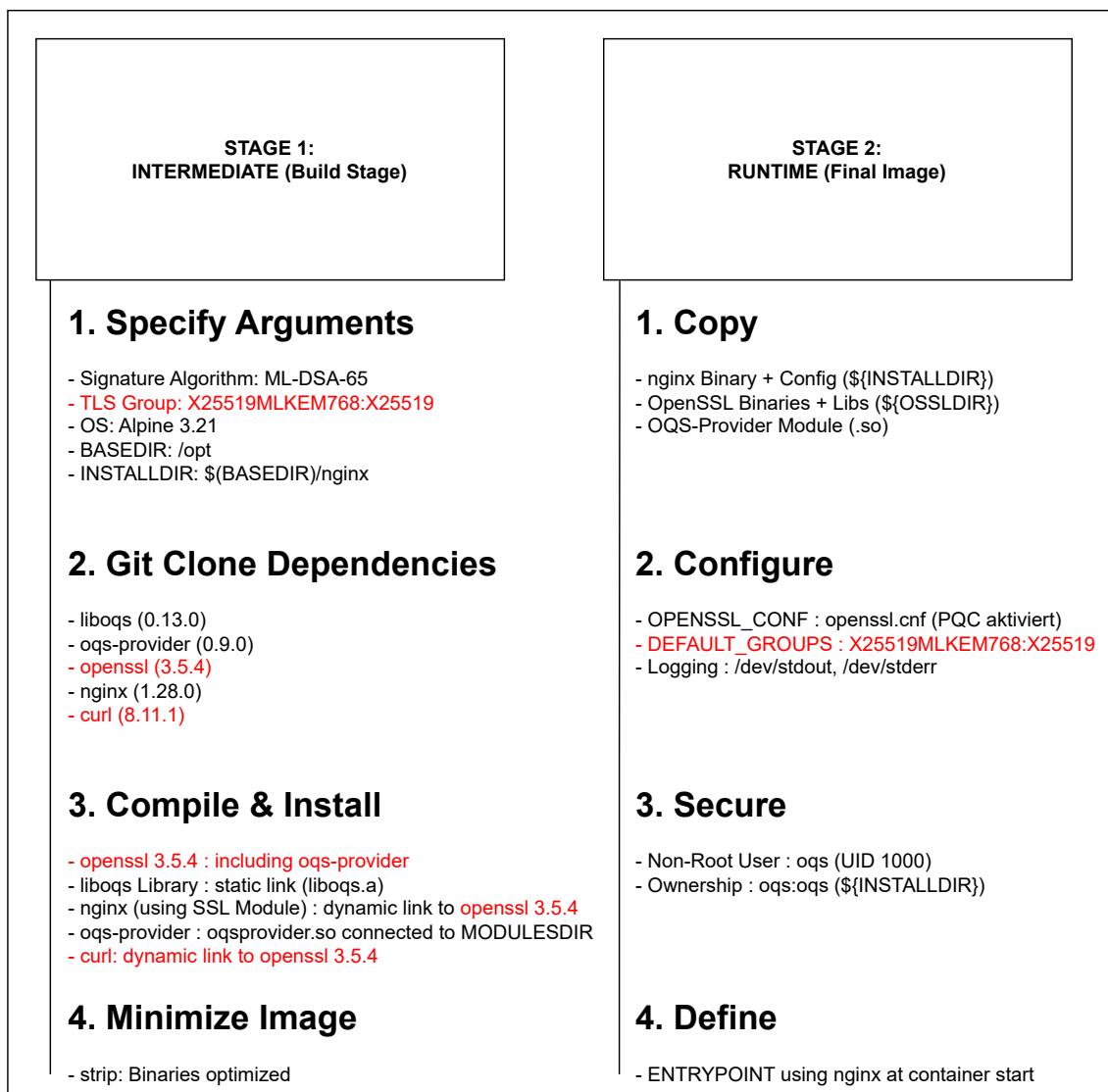
4.5.2 Sidecar-Proxy nginx

Die Implementierung der post-quanten-kryptographischen Absicherung auf Transport-Layer-Ebene basiert auf einer modifizierten Version des nginx-Dockerfiles von open-quantum-safe (2025) sowie spezifischen nginx-Konfigurationsdateien für jeden Sidecar-Proxy. Beide Komponenten realisieren ein konsistentes Konzept bestehend aus Hybrid-Key-Exchange mit ECC als Fallback, welches auf Transport-Layer-Ebene Anwendung findet.

Das modifizierte Dockerfile folgt dem Multi-Stage-Build-Prinzip nach Rosa et al. (2025, S. 1) und integriert OpenSSL 3.5.4, liboqs und den oqs-provider. Abbildung 13 visualisiert die zwei zentralen Phasen (Build-Stage und Runtime-Stage) sowie die Modifikationen gegenüber dem Original-Dockerfile (rote Markierungen). Die Hybrid-Key-Exchange mit ECC-Fallback wird durch die DEFAULT-GROUPS-Konfiguration X25519MLKEM768:X25519 realisiert, eine Konvention, die sich konsistent durch die gesamte Implementierung zieht und in Anhang 3.2.3 ausführlich erläutert wird.

Abbildung 13

Sidecar-Proxy nginx Dockerfile Multi-Stage Build



Anmerkung. Eigene Darstellung.

Die nginx-Konfigurationsdateien implementieren ein standardisiertes Sidecar-Proxy Pattern, welches diese Architektur realisiert. Sie folgen einem konsistenten Schichtenmodell aus globalen Parametern, internen Service-Abstraktionen über Upstream-Blöcke und externen HTTPS-Endpunkten über Server-Blöcke. TLS 1.3 wird durch die Direktive „ssl_ecdh_curve X25519MLKEM768:X25519“ konfiguriert (Listing A-4), wodurch Schlüsselaustausch und Fallback auf klassische ECC-basierte Kurven ermöglicht werden. Die aktivierte Direktive „ssl_protocols TLS 1.3“ wird durch SSL-Zertifikate mit ML-DSA-65-Signaturalgorithmus ergänzt, sodass sowohl der Schlüsselaustausch als auch die Server-Authentifikation post-quantensicher erfolgen. Die gesamte Konfiguration wird in Anhang 3.2.4 erläutert.

4.5.3 DLT-Infrastruktur

Für die PQC-Integration auf Transport-Layer-Ebene wurde die „von-network“-Architektur um einen nginx-PQC-Sidecar-Proxy erweitert. Diese Modifikation stellt die zentrale Anpassung gegenüber dem Original-Quellcode (bcgov, n. d. b) dar und betrifft primär die docker-compose.yml-Konfigurationsdatei (Listing A-5), sowie das Hinzufügen eines neuen Verzeichnisses „pqc_sidecarproxy_nginx“, welches die in Kapitel 4.5.2 und Kapitel 4.5.1 vorgestellten Dockerfile-, nginx-Konfigurationsdatei und Zertifikate für den quantensicheren Sidecar-Proxy enthält. Der Webserver-Container, der im Original-Setup direkt auf Port 9000 exponiert ist (bcgov, n. d. f), verbleibt in der modifizierten Architektur ausschließlich im internen Docker-Netzwerk „von“. Stattdessen terminiert der neu hinzugefügte pqc-sidecarproxy-webserver-Container alle eingehenden TLS-1.3-Verbindungen auf Port 8000 und leitet die Anfragen nach erfolgreicher hybrider Schlüsselvereinbarung und ML-DSA-65-Zertifikatsverifikation als unverschlüsseltes HTTP an den internen Webserver-Container weiter. Die unmodifizierte von-network-Implementierung wird in Anhang 3.2.5 näher erläutert.

Die Integration des Sidecar-Proxies erforderte die Definition eines zusätzlichen, extern zugänglichen Docker-Netzwerks „von_sidecarproxy“, das als gemeinsame Kommunikationsebene für alle PQC-Sidecar-Proxies der Gesamtarchitektur dient. Dieses Netzwerk wird in der docker-compose.yml (Listing A-5) als „external: true“ deklariert. Der pqc-sidecarproxy-webserver-Container ist sowohl mit dem internen von-Netzwerk (für Backend-Kommunikation) als auch mit dem externen von_sidecarproxy-Netzwerk (für Client-Zugriffe) verbunden, wodurch eine strikte Netzwerksegmentierung zwischen interner und externer Kommunikation gewährleistet wird.

4.5.4 Revocation Registry

Für die Integration in die Post-Quantum-gesicherte Gesamtarchitektur wurde der `indy-tails-server` analog zur DLT-Infrastruktur um einen `nginx-PQC-Sidecar-Proxy` erweitert. Diese Modifikation betrifft primär die Docker-Compose-Konfigurationsdatei (Listing A-6), in der ein zusätzlicher Service „`pqc-sidecarproxy-tails-server`“ definiert wurde, sowie das Hinzufügen eines neuen Verzeichnisses „`pqc_sidecarproxy_nginx/`“, welches die in Kapitel 4.5.1 und 4.5.2 vorgestellten Zertifikate, Dockerfile- und `nginx`-Konfigurationsdateien für den quantensicheren Sidecar-Proxy enthält. Der ursprüngliche Tails-Server-Container verbleibt im internen Docker-Netzwerk „`tails-server`“ und exponiert Port 6543 ausschließlich innerhalb dieses Netzwerks. Der neu hinzugefügte `nginx-PQC-Sidecar-Proxy`-Container terminiert alle externen TLS-1.3-Verbindungen auf Port 6543 und leitet die Anfragen nach erfolgreicher hybrider Schlüsselvereinbarung und ML-DSA-65-Zertifikatsverifikation als unverschlüsseltes HTTP an den internen Tails-Server weiter. Die unmodifizierte `indy-tails-server`-Implementierung wird in Anhang 3.2.6 näher erläutert.

Die Netzwerk-Integration folgt dem etablierten Sidecar-Proxy Pattern. Der „`pqc-sidecarproxy-tails-server`“-Container ist sowohl mit dem internen „`tails-server`“-Netzwerk (für Backend-Kommunikation) als auch mit dem externen „`von_sidecarproxy`“-Netzwerk (für Client-Zugriffe) verbunden. Diese Dual-Network-Architektur erzwingt, dass alle externen Zugriffe auf den Tails-Server über den quantensicheren Sidecar-Proxy geleitet werden.

4.5.5 SSI-Agenten

Die SSI-Agent-Schicht bildet die Anwendungsebene der Gesamtarchitektur und implementiert die drei klassischen Rollen des SSI-Ökosystems Issuer, Holder und Verifier (Kapitel 2.1) samt ihrer dedizierten `nginx-PQC-Sidecar-Proxies`. Die Architektur folgt dem in Kapitel 5.2.2 beschriebenen Referenzmodell. Alle drei Agenten basieren auf dem unmodifizierten ACA-Py-Base-Image (Listing A-7) und werden ausschließlich über Kommandozeilen-Parameter konfiguriert, ohne Änderungen am ACA-Py-Quellcode vorzunehmen.

Listing A-8 zeigt die Docker-Compose-Konfiguration der drei ACA-Py-Agenten innerhalb der Gesamtarchitektur. Die Agent-Konfiguration erfolgt hierbei vollständig deklarativ über Docker-Compose-Service-Definitionen, die jeweils den „`start`“-Befehl von ACA-Py mit rollenspezifischen Parametern aufrufen. Jeder Agent wird in einem dedizierten Docker-Netzwerk isoliert betrieben und über einen `nginx-PQC-Sidecar-Proxy` mit quantensiche-

rer TLS-1.3-Verschlüsselung nach außen exponiert. Die Wallet-Konfiguration nutzt persistente Docker-Volumes zur Speicherung von DID, Credentials und Connections über Container-Neustarts hinweg. Die Agents verbinden sich über PQC-gesicherte Sidecar-Proxies mit der DLT-Infrastruktur und der Revocation Registry. Auto-Response-Features ermöglichen vollständig scriptgesteuerte SSI-Workflows ohne manuelle Interaktion. Die Netzwerk-Architektur folgt einem strikten Isolation-Prinzip, sodass Agents nur über das externe „von_sidecarproxy“-Netzwerk untereinander kommunizieren können. Health-Checks und Service-Dependencies orchestrieren deterministische Startup-Sequenzen und eliminieren Race-Conditions während des Deployments. Eine ausführliche Darstellung der Agent-Konfigurationsparameter und der Sidecar-Proxy-Architektur findet sich in Anhang 3.2.8.

4.5.6 Docker Orchestrierung der Gesamtarchitektur

Die in den vorangegangenen Abschnitten beschriebenen Einzelkomponenten – Zertifikatsinfrastruktur, PQC-Sidecar-Proxies, DLT-Infrastruktur, Revocation Registry und SSI-Agenten – werden mittels einer mehrstufigen Docker-Compose-Orchestrierung zu einem funktionsfähigen Gesamtsystem integriert. Der Startprozess der Gesamtarchitektur folgt einer deterministischen Sequenz, die in Listing A-9 dokumentiert ist und die korrekten Abhängigkeiten zwischen den Infrastrukturschichten gewährleistet.

Die Orchestrierung gliedert sich in drei sequenzielle Phasen, die jeweils durch separate Docker-Compose-Konfigurationen gesteuert werden. In der ersten Phase wird die DLT-Infrastruktur über das von-network-manage-Skript (Listing A-10) initialisiert, wodurch die vier Indy-Validator-Nodes, der Genesis-Webserver sowie der zugehörige nginx-PQC-Sidecar-Proxy gestartet werden. Diese Phase erzeugt das gemeinsame externe Docker-Netzwerk „von_sidecarproxy“, welches als zentrale Kommunikationsschicht für alle quantensicheren Verbindungen dient. Die zweite Phase umfasst die Initialisierung der Revocation Registry mittels des indy-tails-server-manage-Skripts (Listing A-11), wodurch der Tails-Server-Container sowie dessen nginx-PQC-Sidecar-Proxy-Frontend bereitgestellt werden. In der dritten Phase werden schließlich die SSI-Agenten gemeinsam mit ihren jeweiligen PQC-Sidecar-Proxies über die projektspezifische Docker-Compose-Konfiguration (Listing A-8) gestartet.

Die in Abbildung 14 visualisierte Containerarchitektur verdeutlicht die resultierende Topologie des Systems. Die Gesamtarchitektur umfasst insgesamt 14 Container, die sich auf die drei funktionalen Schichten verteilen. Die DLT-Schicht besteht aus insgesamt sechs

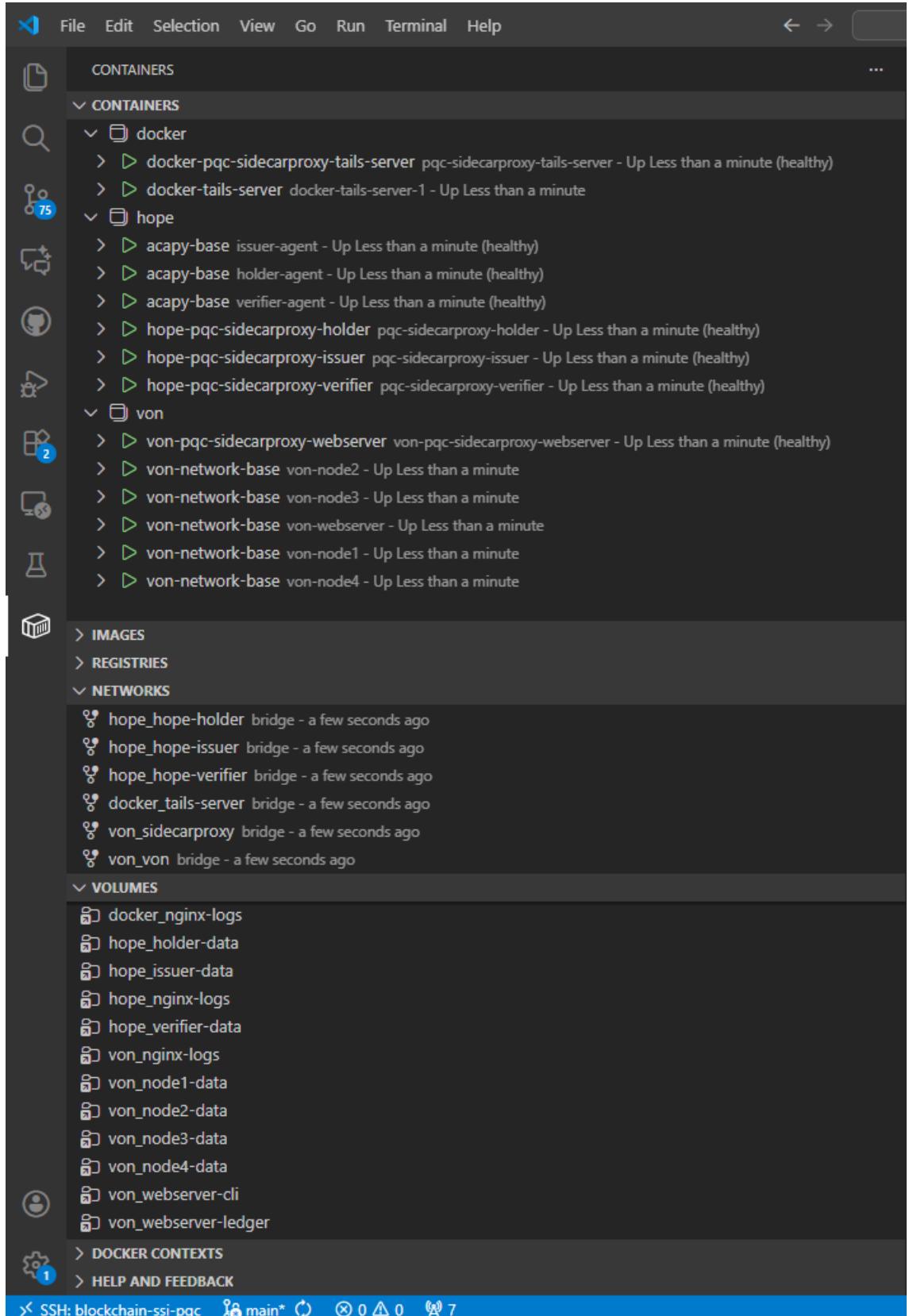
Containern, den vier Validator-Nodes, einem Webserver und einem nginx-PQC-Sidecar-Proxy. Die Revocation-Schicht umfasst insgesamt zwei Container, einen Tails-Server und dessen nginx-PQC-Sidecar-Proxy. Die Agent-Schicht besteht aus insgesamt sechs Containern für die drei SSI-Agenten und ihre jeweiligen PQC Sidecar-Proxies.

Die Netzwerktopologie nutzt sechs dedizierte Docker-Bridges, um die logische Separation der Schichten zu realisieren. Das „hope_hope-holder“-, „hope_hope-issuer“- und „hope_hope-verifier“-Netzwerk verbinden jeweils die SSI-Agenten mit der DLT-Schicht. Das „docker_tails-server“-Netzwerk isoliert die Revocation-Infrastruktur, während das „von_sidecarproxy“-Netzwerk die PQC-Sidecar-Proxies miteinander verbindet. Das „von_von“-Netzwerk integriert die Validator-Nodes untereinander und mit der Revocation-Schicht.

Die Datenpersistenz wird durch elf Docker Volumes realisiert. Während „docker_nginx-logs“ und „von_nginx-logs“ die Webserver-Logs aggregieren, speichern „hope_holder-data“, „hope_issuer-data“ und „hope_verifier-data“ die lokalen Wallets und kryptografischen Materialien der SSI-Agenten. Die Validator-Node-Datenbank wird durch „von_node1-data“ bis „von_node4-data“ persistiert, während „von_webserver-cli“ und „von_webserver-ledger“ die CLI-Konfigurationen und Ledger-Zustandsdaten verwalten. Diese Volumenstruktur entkoppelt den Containern-Lebenszyklus von der Persistenz der Daten und ermöglicht die Wiederaufnahme der Systemzustände über Container-Neustarts hinweg.

Die „depends_on“-Direktiven in den Docker-Compose-Konfigurationen (Listing A-5, Listing A-6 und Listing A-8) definieren explizite Startup-Abhängigkeiten, die über reine Ausführungsreihenfolgen hinausgehen. Ohne Synchronisation führen verfrühte Interaktionen abhängiger Services zu „faulty interactions“ während der Boot-Phase (De Iasio & Zimeo, 2021, S. 25). Die implementierte Konfiguration eliminiert diese Race-Conditions, indem sie sicherstellt, dass die nginx-PQC-Sidecar-Proxies nicht gestartet werden, bevor die ihnen zugeordneten Backend-Services den Status („ready“) erreichen. Diese Orchestrierung erzwingt eine deterministische Startup-Sequenz, die verhindert, dass abhängige Komponenten auf Dienste zugreifen, die sich zwar im Status „Running“, aber noch nicht im Status „Ready“ befinden (De Iasio & Zimeo, 2021, S. 29).

Abbildung 14
Docker-Compose-Übersicht der ersten Iteration



Anmerkung. Eigene Darstellung.

4.6 Formative Evaluation

In Übereinstimmung mit dem in Kapitel 3.3.4 entworfenen Evaluationsdesign setzt dieser Abschnitt die erste definierte Evaluationsepisode (Tabelle 3) um. Charakterisiert als formative und artifizielle Untersuchung, liegt der Fokus dieser Phase exklusiv auf der Validierung der TLS sowie der korrekten Konfiguration der Infrastrukturkomponenten. Ziel ist es, gemäß der Prinzipien von Venable, Pries-Heje und Baskerville (2016, S. 82–83) Designfehler in der Sidecar-Proxy-Architektur frühzeitig zu identifizieren.

Methodisch erfolgt dies primär durch White-Box-Tests. Da bei diesem Verfahren die internen Strukturen und Implementierungsdetails der Software bekannt sind und gezielt in die Prüfung einbezogen werden (Myers, Sandler & Badgett, 2012, S. 10), eignet es sich besonders gut, um die korrekte Konfiguration der kryptographischen Primitive innerhalb der Container-Architektur durch eine detaillierte Analyse der Handshake-Logs zu validieren.

Für die technische Durchführung dieser Analysen war die Bereitstellung eines PQC-fähigen Browsers zwingend erforderlich, da aktuelle Produktivbrowser noch keine PQC-Unterstützung in ihren TLS-Implementierungen bieten. Diese Limitation führt bei Verbindungsversuchen zu PQC-fähigen Servern unweigerlich zu einem Cipher-Mismatch, wie in Abbildung A-7 veranschaulicht. Um diese Inkompatibilität zu überwinden, wurde ein Chromium-basierter Browser mit integrierter PQC-Unterstützung kompiliert (Anhang 3.3.2). Dieser ermöglicht die Durchführung von TLS-Handshakes mit hybriden sowie rein PQC-basierten Algorithmen und dient als fundamentales Werkzeug für die experimentelle Analyse der implementierten Verfahren.

4.6.1 Validierung der Zertifikatskette und ML-DSA-Signaturen

Zur Verifikation der kryptographischen Integrität der implementierten Public Key Infrastructure (PKI) wurde die Zertifikatskette der Sidecar-Proxies mittels openssl-Diagnosewerkzeugen analysiert. Ziel war der Nachweis, dass die ausgelieferten X.509-Zertifikate korrekt auf den spezifizierten Post-Quanten-Signaturalgorithmen basieren. Die Inspektion des vom Issuer-Agenten-Proxy bereitgestellten Zertifikats (Abbildung A-8) bestätigt, dass der Public Key des Zertifikats („pqc reverseproxy issuer agent“) den Algorithmus ML-DSA-65 verwendet. Des Weiteren belegt der Signaturalgorithmus ML-DSA-87, dass die Zertifikatskette valide durch die „Master Thesis PQC Root CA“ signiert wurde, was die erfolgreiche Generierung und Einbindung der Dilithium-basierten Zertifikate in den TLS-Handshake beweist.

4.6.2 Validierung der TLS 1.3 Algorithmen-Aushandlung

Die erfolgreiche Integration der PQC-Algorithmen in das Transportprotokoll wurde durch einen Verbindungsaufbau mittels „openssl s_client“ verifiziert. Wie in Abbildung A-9 dargestellt, konnte erfolgreich eine TLS-1.3-Sitzung etabliert werden. Die Analyse der Handshake-Parameter bestätigt die Verwendung der hybrid-post-quanten Schlüsselaustauschgruppe „X25519MLKEM768“, welche den klassischen elliptischen Kurvenalgorithmus X25519 mit dem KEM-Verfahren ML-KEM-768 kombiniert. Zudem wird für die Authentifizierung des Peer-Zertifikats der Signaturalgorithmus ML-DSA-65 ausgewiesen. Diese Ergebnisse validieren die korrekte Konfiguration der Open Quantum Safe (OQS)-Provider-Bibliothek innerhalb der Proxy-Komponenten und belegen die praktische Funktionsfähigkeit des hybriden Schlüsselaustauschs im Zusammenspiel mit PQC-Signaturen.

4.6.3 Validierung der Ledger-Initialisierung

Die operative Funktionsfähigkeit des Hyperledger Indy Netzwerks wurde primär über das Web-Interface des Blockchain-Servers validiert. Wie in Abbildung A-10 dargestellt, zeigen die Statusindikatoren aller vier Validator-Nodes eine aktive Beteiligung am Konsensus-Protokoll (Status Node1-4), womit der Distributed Ledger erfolgreich initialisiert ist. Simultan belegt diese Abbildung die korrekte PQC-Absicherung der Webserver-Komponente. Der Zugriff erfolgt über den eigens kompilierten PQC-Chromium-Browser, dessen Security-Panel explizit eine authentifizierte TLS-1.3-Verbindung unter Verwendung der hybriden Schlüsselaustauschgruppe „X25519MLKEM768“ ausweist.

Als zweite notwendige Bedingung für die spätere Anbindung der SSI-Agenten wurde die Verfügbarkeit der Genesis-Datei verifiziert. Abbildung A-11 dokumentiert den Abruf des Endpunkts „/genesis“ mittels „curl“. Die erfolgreiche Rückgabe der JSON-formatierten Genesis-Transaktionen bestätigt, dass die für die Anbindung externer Clients erforderlichen Netzwerkinformationen korrekt publiziert werden.

4.6.4 Validierung der ACA-Py API-Verfügbarkeit

Die funktionale Erreichbarkeit der SSI-Agenten wurde durch eine systematische Analyse der Initialisierungsphase und der anschließenden API-Verfügbarkeit validiert. Die Logging-Ausgabe des Issuer-Agenten (Listing A-12) dokumentiert den erfolgreichen Abruf der Genesis-Datei und die vollständige Ledger-Konfiguration. Die Erstellung eines neuen Wallet-Profils mit Askar-Backend und die erfolgreiche Initialisierung des Inbound- und

Outbound-Transports demonstrieren die korrekte Konfiguration des ACA-Py-Agents. Die durchgeführten Health-Checks über den Endpunkt „/status/ready“ bestätigen die vollständige Initialisierung und Bereitschaft des Agenten.

Die Visualisierung der Swagger-basierten Admin-Oberfläche (Abbildung A-12) ergänzt diese technischen Log-Daten durch den Nachweis, dass die Admin-API über den nginx-PQC-Sidecar-Proxy fehlerfrei erreichbar ist und alle administrativen Endpunkte zur Steuerung der Agenten-Komponente bereitstellt. Die Tatsache, dass die Swagger-Oberfläche unter dem PQC-gesicherten HTTPS-Endpoint vollständig funktionsfähig ist, belegt die korrekte TLS-Terminierung am Proxy sowie die fehlerfreie Weiterleitung der HTTP-Anfragen an den ACA-Py-Container.

4.6.5 Validierung der Netzwerkisolation

Die integrale Sicherheitseigenschaft der Netzwerksegmentierung wurde durch eine Inspektion der Docker-Netzwerktopologie und systematische Erreichbarkeitstests validiert. Abbildung A-13 belegt diese Topologie anhand des „docker network inspect“-Outputs. Der Issuer-Agent befindet sich exklusiv im Netzwerksegment „hope_hope-issuer“, während der Holder-Agent im Segment „hope_hope-holder“ isoliert ist. In jedem dieser Segmente fungiert der zugehörige nginx-PQC-Sidecar-Proxy als einziger Ingress-Punkt.

Abbildung A-14 demonstriert die Wirksamkeit dieser Isolation auf zwei Ebenen. Erstens zeigt die Prozessliste („docker ps“), dass lediglich die Sidecar-Proxies externe Ports (8020, 8030, 8040) an das Host-System binden, während die Ports der ACA-Py-Container nicht exponiert sind. Zweitens beweisen die Inter-Container-Verbindungstests die logische Trennung. Ein direkter Zugriffsversuch aus dem „issuer-agent“-Container auf den „holder-agent“-Container schlägt mit einem DNS-Auflösungsfehler („Could not resolve host“) fehl, da kein Routing zwischen den isolierten Netzwerkbrücken existiert. Im Gegensatz dazu ist der lokale Zugriff des „pqc-sidecarproxy-holder“ auf seinen zugehörigen Agenten erfolgreich möglich.

4.7 Erkenntnisse und Anpassungsbedarfe

Die erste Iteration bildet das technologische Fundament der Forschungsarbeit. Die formative Evaluation (Kapitel 4.6) validierte die operative Integrität der entwickelten Architektur. Die verteilten Micro-Services, der Hyperledger Indy Ledger und die PQC-Sidecar-Proxies interagieren funktional korrekt. Diese Initialphase generierte jedoch spezifische Erkenntnisse, die eine gezielte Weiterentwicklung in der zweiten Iteration motivieren. Diese werden nachfolgend in Bezug auf die Designziele analysiert.

4.7.1 Abgleich mit den Designzielen und kritische Erkenntnisse

Das primäre Designziel, die Absicherung der Transportebene in einem SSI-Ökosystem mittels PQC, wurde vollständig erreicht. Die erfolgreiche Validierung des Sidecar-Proxy-Pattern belegt die Machbarkeit einer transparenten PQC-Migration für Legacy-Systeme ohne Eingriffe in deren Kerncode. Die implementierte Micro-Segmentation erfüllt zudem die architektonischen Anforderungen an logische Netzsegmentierung in KRITIS-Umgebungen (Tabelle 5).

Durch die strikte Entkopplung der kryptographischen Terminierung von der Business-Logik durch die Sidecar-Proxy-Architektur konnte eine PQC-Integration realisiert werden, die die Kernprozesse der Identitätsverwaltung funktional nicht beeinträchtigt. Diese architektonische Entscheidung ermöglicht es, sicherheitskritische Updates an der Krypto-Komponente vorzunehmen, ohne die Integrität der komplexen SSI-Logik zu gefährden. Die empirische Validierung der Zertifikatsketten und TLS-1.3-Handshake-Protokolle (Kapitel 4.6) bestätigt die technische Reife dieser Architekturentscheidung.

Bezüglich der Algorithmenauswahl und Sicherheitsbewertung operationalisiert die erste Iteration die in Tabelle 5 definierten BSI-Vorgaben für PQC durch die Implementierung der NIST-standardisierten Algorithmen ML-DSA-65 und ML-KEM-768, welche explizit die NIST Security Strength Categories 3/5 erfüllen (BSI, 2025b, Kap. 2.4, 5.3.4.2). Die formative Evaluation bestätigte die technische Machbarkeit dieser Algorithmen in der Sidecar-Proxy-Infrastruktur durch die erfolgreiche Validierung hybrider Zertifikatsketten und TLS-1.3-Handshake-Protokolle mit der Schlüsselaustauschgruppe „X25519MLKEM768“.

Bezüglich der kryptografischen Agilität zeigt die erste Iteration, dass das Designziel einer architektonischen Vorbereitung auf Algorithmenaustauschbarkeit erreicht wurde. Die Container-basierte Sidecar-Proxy-Architektur ermöglicht es, kryptografische Bibliotheken durch Rolling Updates der Proxy-Images auszutauschen, ohne ACA-Py oder die übrige

Systemlogik anzupassen, was direkt die von Mehrez und El Omri (2018, S. 102) geforderte „Extensibility“ kryptoagiler Systeme adressiert. Gleichzeitig nutzt die TLS-1.3-Integration eine konfigurationsbasierte Fallback-Chain („DEFAULT_GROUPS:X25519:ML-KEM-768:mlkem768x25519:mlkem1024“), sodass hybride und klassische Verfahren orthogonal ausgehandelt und bei Inkompatibilitäten automatisch gewechselt werden können (Rescorla, 2018, S. 26). Diese Protokoll- und Infrastrukturmechanismen realisieren damit zentrale Kryptoagilitäts-Eigenschaften wie „Fungibility“ und „Updateability“ (Mehrez & El Omri, 2018, S. 102–103) und entsprechen dem Erfordernis, kryptografische Komponenten ohne grundlegende Systemumgestaltung migrieren zu können (Kreutzer, Wolf & Waidner, 2024, S. 670).

Eine zentrale Limitation identifizierte jedoch die Analyse der Sicherheitsmodelle. Während die Transportebene durch die Sidecar-Proxies vollständig quantensicher abgesichert ist (Data-In-Motion), verbleiben VC und DID-Dokumente (Data-At-Rest) mittels klassischer Kryptografie verschlüsselt. Diese Diskrepanz zwischen Transportschutz (PQC-gesichert) und Datenpersistierung (klassische Kryptografie) widerspricht dem mehrschichtigen Sicherheitsansatz „Defense in Depth“ von Alsaqour et al. (2021, S. 242–243), bei dem konsistente, sich gegenseitig verstärkende Kontrollen auf mehreren Ebenen implementiert werden, um Ressourcen und Assets umfassend zu schützen. Diese Erkenntnisse folgern eine Erweiterung des Sicherheitsmodells von der Transportebene auf die Applikationsschicht in der zweiten Iteration.

4.7.2 Design-Refinements und Operationalisierung der zweiten Iteration

Die systematische Analyse der Evaluationsergebnisse führt zu zwei konvergenten Design-Refinements, die die identifizierten Sicherheitslücken adressieren und die zweite Iteration strukturieren.

Refinement 1: Integration der PQC auf Applikationsebene. Dieses Refinement adressiert die identifizierte Sicherheitslücke durch technische Erweiterung der PQC-Integration von der Transportebene auf die Applikationsebene. Die direkte Einbindung der liboqs-Bibliothek in die ACA-Py-Agenten ermöglicht ML-DSA-65-Signaturen für VC und DIDComm-Nachrichten, wodurch die kritische Diskrepanz zwischen quantensicherer Transportverschlüsselung und ungeschützter Datenpersistierung geschlossen wird. Die Implementierung als modulares Plugin-System folgt dem Stabilitätsprinzip nach Hamza (2005, S. 1–2), indem kryptografische Primitive in stabile Abstraktionsschichten (Kryptografie-Wrapper, DID-Verarbeitung, Integration-Patching) gekapselt werden, während instabile Algorithmen-spezifische Parameter (ML-DSA-65, ML-KEM-768) als aus-

tauschbare Konfigurationselemente behandelt werden. Dies realisiert das Open-Closed-Prinzip nach Martin (2003) und minimiert den Aufwand für zukünftige Reseparation bei Algorithmen-Updates, da nur die Konfigurationsebene modifiziert werden muss, ohne die stabile Plugin-Architektur zu verändern.

Refinement 2: Hybride Sicherheitsarchitektur mit redundanter Tiefenstaffelung. Dieses Refinement etabliert ein übergeordnetes „Defense in Depth“-Modell nach Alsaqour et al. (2021, S. 242–243) durch beibehaltene Sidecar-Proxies als erste Verteidigungslinie (TLS) bei gleichzeitiger Quantensicherung der Datenobjekte auf Anwendungsebene (Application Layer Security). Diese redundante Absicherung schafft eine tiefengestaffelte Sicherheitsarchitektur, bei der ein Bruch einzelner Schichten, nicht automatisch zum Kollaps der Gesamtsicherheit führt. Das Refinement implementiert damit die von Alsaqour et al. (2021, S. 242–243) geforderte Prinzipienkonsistenz über alle Architekturebenen hinweg und gewährleistet die Resilienz des Systems gegen hybride Angriffszenarien.

5 Zweite Iteration der Artefaktentwicklung

5.1 Designziele dieser Iteration

Die zweite Iteration der Artefaktentwicklung baut auf der innerhalb der ersten Iteration erfolgreich validierten Basisarchitektur auf und korrespondiert erneut mit der DSRM-Phase zwei Objectives nach Peffers et al. (2007, S. 54). Der Fokus dieser Iteration liegt auf der Erweiterung des Prototypen um eine tiefgreifende PQC-Integration auf der Anwendungsebene (Application Layer). Im Kontext des Drei-Zyklen-Modells nach Hevner (2007, S. 88) wird der Design Cycle intensiviert, um die kryptografische Sicherheit von der reinen Transportsicherung (TLS 1.3) auf die tatsächlichen Nutzdaten (VC und DID-Dokumente) auszuweiten und somit eine Ende-zu-Ende-Sicherheit zu gewährleisten.

Die Designziele dieser Iteration leiten sich konsistent aus den in Kapitel 1.3 definierten Forschungsfragen ab, wobei eine inhaltliche Vertiefung der technischen Anforderungen erfolgt.

Bezüglich FF1 (Systemarchitektur & Compliance) wird das Ziel verfolgt, die SSI-Kernprozesse so zu modifizieren, dass sie quantenresistente Signaturen und Schlüsselformate nativ unterstützen. Das Design muss sicherstellen, dass die Unveränderlichkeit und Authentizität von Identitätsnachweisen unabhängig vom Transportkanal auch langfristig gegenüber Quantencomputer-Angriffen gewährleistet bleibt, was eine zentrale Anforderung für den Einsatz in KRITIS-Umgebungen darstellt (BSI, 2025b, S. 25).

Hinsichtlich FF2 (Algorithmenauswahl & Sicherheitsbewertung) wird das Ziel verfolgt, die praktische Machbarkeit von NIST-standardisierten Post-Quantum-Algorithmen in den Kernkomponenten des SSI-Systems nachzuweisen. Der Fokus liegt auf der Integration quantenresistenter Signatur- und Verschlüsselungsverfahren auf der Applikationsebene, um die digitale Authentizität und Integrität von Identitätsnachweisen langfristig gegen Quantencomputer-Angriffe zu schützen.

Für FF3 (Kryptografische Agilität) zielt diese Iteration auf die Implementierung von Agilitätsmechanismen direkt in den Datenstrukturen ab. Das System soll so gestaltet werden, dass es hybride Szenarien unterstützt und eine Koexistenz sowie den nahtlosen Wechsel zwischen klassischen und post-quanten Kryptografieverfahren innerhalb der DID-Methoden und Credential-Definitionen ermöglicht, ohne die Interoperabilität grundlegend zu gefährden.

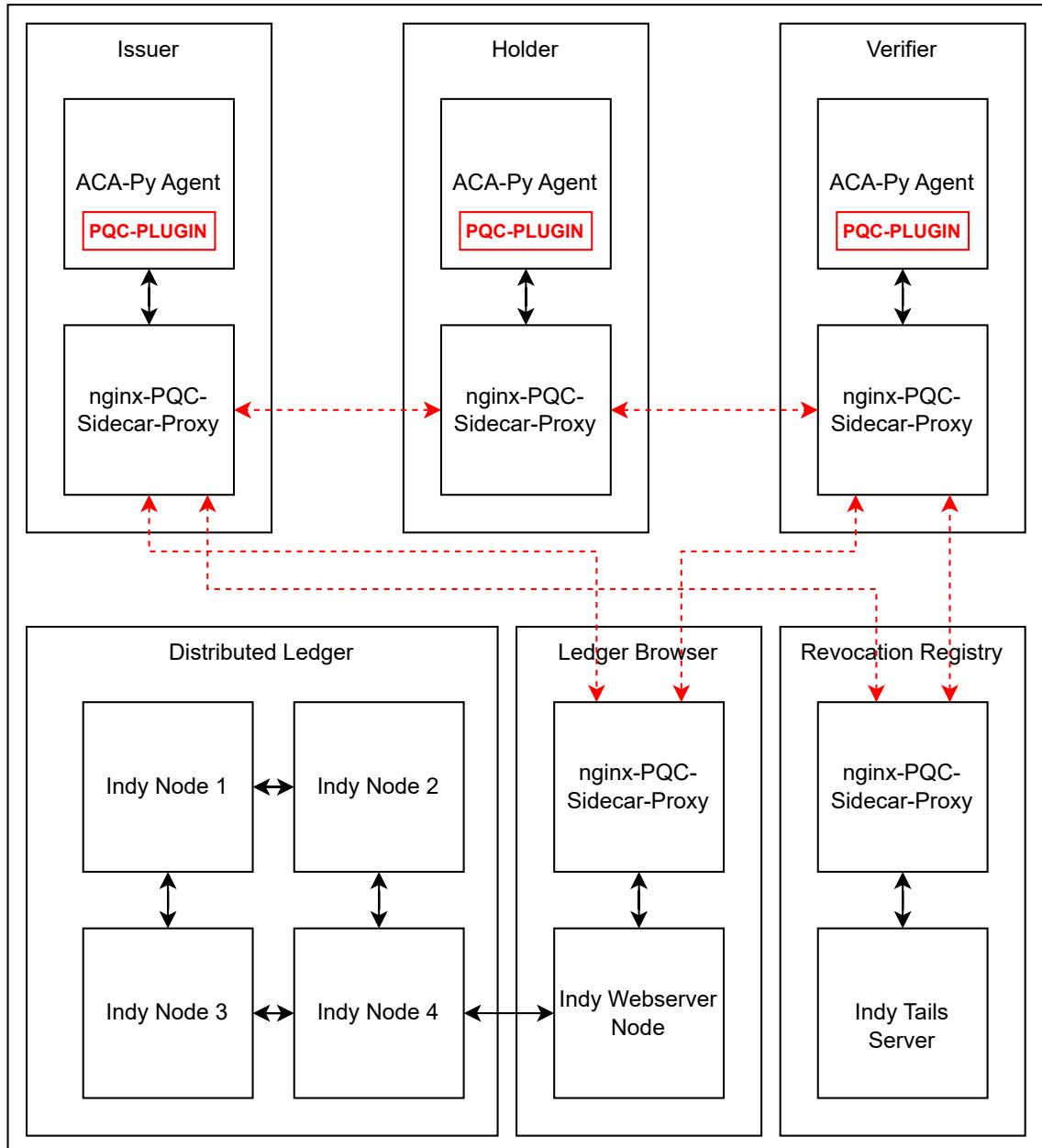
5.2 Architekturentwurf

5.2.1 Gesamtarchitektur

Die Gesamtarchitektur der zweiten Iteration erweitert die in Kapitel 4.4.1 etablierte dreischichtige Containerarchitektur um eine zusätzliche Kryptoebene auf der Anwendungsschicht. Während die in der ersten Iteration implementierte Sidecar-Proxy-Architektur mit TLS 1.3 und hybrider Schlüsseleinigung (X25519 + ML-KEM-768) vollständig beibehalten wird und weiterhin die Transportverschlüsselung zwischen den Komponenten gewährleistet, wird in dieser Iteration die kryptografische Absicherung auf die SSI-Agenten-Schicht ausgeweitet.

Abbildung 15 visualisiert diese Erweiterung durch die Integration von „PQC-PLUGIN“-Modulen in die drei ACA-Py-Instanzen (Issuer, Holder, Verifier). Diese Plugin-Module ermöglichen die Verwendung quantenresistenter Signaturalgorithmen (ML-DSA-65) innerhalb von VC und DID sowie die Verwendung quantenresistenter Schlüsselkapselungsverfahren (ML-KEM-768) für die DIDComm-Messaging-Verschlüsselung. Durch diese duale Architektur mit Sidecar-Proxies für die Transportebene und Plugin-Module für die Applikationsebene wird eine durchgängige Ende-zu-Ende-Quantenresistenz realisiert, die sowohl Data-In-Motion als auch Data-At-Rest schützt. Die DLT-Infrastruktur (vier Hyperledger-Indy-Validator-Nodes, Ledger-Browser, Webserver), die Revocation Registry (Tails-Server) sowie die zugrundeliegende Docker-Netzwerktopologie bleiben strukturell unverändert und gewährleisten die Kontinuität der in der ersten Iteration validierten Infrastrukturkomponenten.

Abbildung 15
Gesamtarchitekturentwurf der zweiten Iteration



Anmerkung. Eigene Darstellung basierend auf Abbildung 10.

5.2.2 ACA-Py Applikationsarchitektur

Während Kapitel 4.4.2 die klassische Applikationsarchitektur als geschichtete, unveränderliche Referenzimplementierung vorstellt, demonstriert die zweite Iteration die transparente Applikationsschicht-Integration von PQC durch ein modulares Plugin-System, das bestehendes Kerncode-Design respektiert und durch gezieltes Patching erweitert. ACA-Py-Plugins ermöglichen hierbei eine standardisierte Erweiterbarkeit, ohne die ACA-Py-Codebasis zu überlasten („ACA-Py Plugins - ACA-Py Docs“, n. d.).

Abbildung 16 visualisiert diese erweiterte Architektur und verdeutlicht die Integration des PQC-Plugins als zentrale Interceptions- und Delegationsschicht. Die roten Markierungen heben die Unterschiede zur klassischen ACA-Py Applikationsarchitektur (Abbildung 11) hervor. Das Plugin ist seitlich an alle vier klassischen Schichten (Protocol Handlers, Wallet Interface, Transport Layer, externe Business Logic) angebunden, woraus sich bidirektionale Pfeile ergeben. Diese Bidirektionalität symbolisiert die transparente Interception. Wenn eine klassische Schicht eine Operation initiiert, wird diese zunächst vom Plugin abgefangen, eine PQC-äquivalente Operation durchgeführt und das Ergebnis zurückgegeben. Diese Interceptions-Delegation ermöglicht es dem Plugin, als Zwischenschicht zu fungieren, ohne die übergeordnete Geschäftslogik zu modifizieren.

Um langfristige Wartbarkeit und Kryptoagilität zu sichern, folgt das PQC-Plugin intern dem Ansatz der Separation of Concerns for Evolving Systems nach Hamza (Hamza, 2005, S. 1–2). Das Plugin ist nach dem Software Stability Model in drei internen Schichten strukturiert, denen nachfolgend die konkreten Implementierungskomponenten zugeordnet werden.

Die stabile Kern-Ebene, das Enduring Business Theme (EBT), kapselt die Enduring Business Knowledge der Kryptografie. Sie umfasst abstrakte Operationen wie Signaturerstellung, Schlüsselgenerierung, Schlüsselaustausch und Verschlüsselung. Diese Operationen sind unabhängig vom konkreten Algorithmus und stellen die stabile Kern-EBT dar. Wenn zukünftig ML-DSA-87 statt ML-DSA-65 verwendet wird, bleibt diese Abstraktionsschicht unverändert, es ändert sich nur die konkrete Implementierung. Nach Hamza sind EBTs die Basis für Stabilität, da ein System ohne EBTs bei Änderungen neu separiert werden muss (Hamza, 2005, S. 1–2).

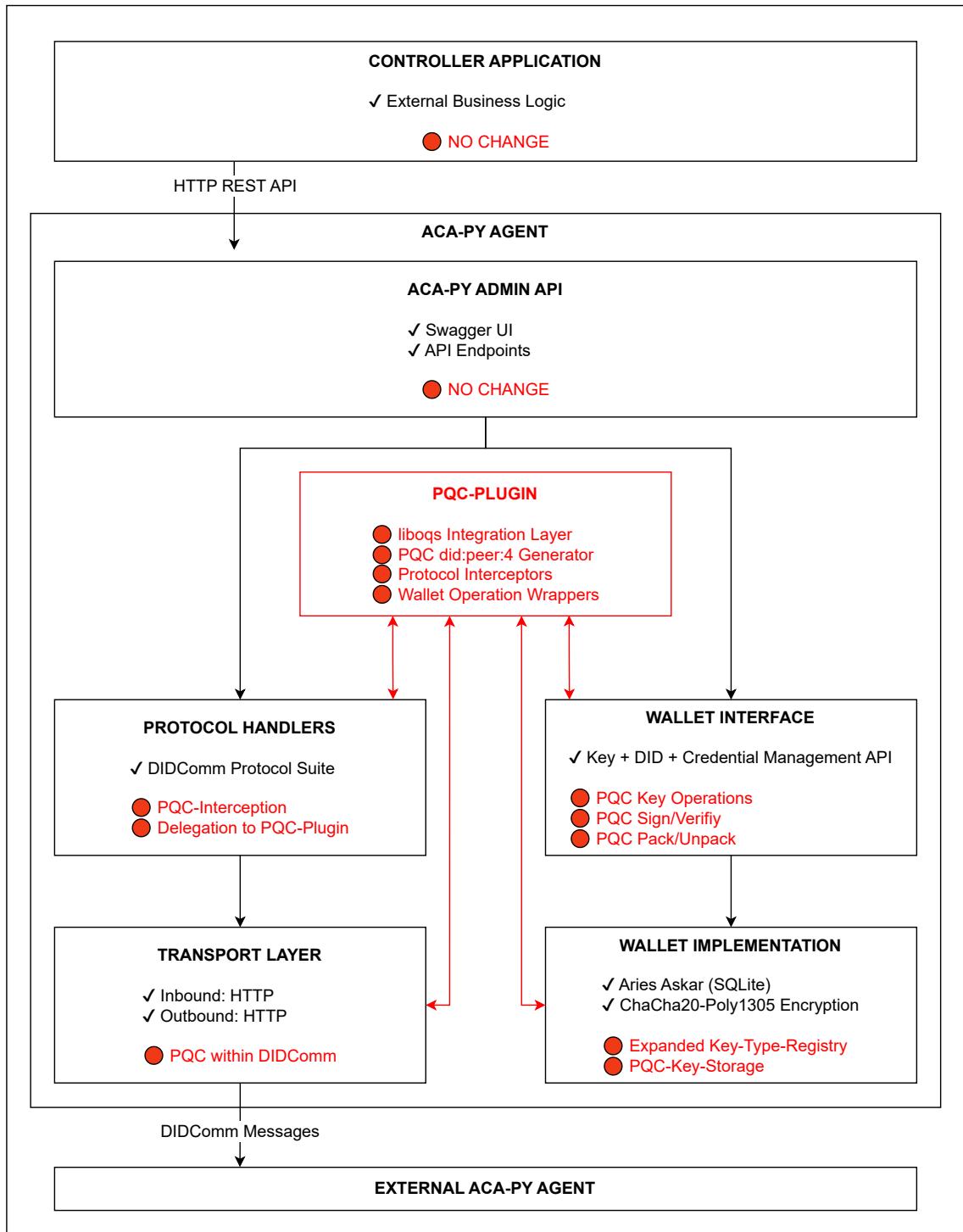
Konkret wird die Protocol Handlers-Schicht der ACA-Py-Architektur erweitert. Während die klassische Architektur DID Exchange, Credential Issuance und Presentation Proof nur mit klassischen Signaturen durchführt, delegiert das Plugin diese Operationen und nutzt ML-DSA-65 statt ed25519 (openwallet-foundation, n. d. l). Die abstrakte Signatur-Operation als EBT bleibt gleich. Nur der Algorithmus (ML-DSA-65 statt ed25519) ändert sich.

Die semi-stabile Geschäfts-Objekt-Schicht, das Business Object (BO), konkretisiert die abstrakten EBT-Operationen in praktische Strukturen. BOs sind nach Hamza intern adaptierbar durch sogenannte hooks, aber extern stabil (Hamza, 2005, S. 1083). Im PQC-Plugin sind die BOs die DID-Verarbeitung mit W3C-konformen Strukturen, die Multicodec-Identifikatoren und die Key-Type-Registry mit ML-DSA-65 und ML-KEM-768. Bei zukünftiger Migration zu ML-KEM-1024 können neue Multicodec-Präfixe hinzugefügt werden, ohne die DIDComm-Schnittstellen zu brechen.

Konkret wird die Wallet Interface-Schicht nicht ersetzt, sondern erweitert. Das Aries-Askar-Backend bleibt funktionsfähig und verwaltet weiterhin klassische Kryptografie (openwallet-foundation, n. d. l.). Das Plugin registriert zusätzlich neue Operationsmethoden wie ML-DSA-65-Signaturen, ML-KEM-768-Key-Encapsulation sowie Speicherfunktionen für größere Schlüssellängen in SQLite mit ChaCha20-Poly1305-Verschlüsselung. Die Key-Type-Registry und DIDComm-Verschlüsselung werden ebenfalls dynamisch erweitert. Das Plugin registriert ML-DSA-65 und ML-KEM-768 als vollwertige Schlüsseltypen neben den klassischen Varianten ed25519 und x25519 (openwallet-foundation, n. d. l.). Die DIDComm-Verschlüsselung wird orthogonal erweitert, sodass PQC-Nachrichten ML-KEM-768-Key-Encapsulation nutzen. Diese Erweiterungen verdeutlichen die BO-Struktur, indem neue Algorithmen intern hinzugefügt werden, während die externen Schnittstellen stabil bleiben.

Die volatile Implementierungs-Ebene, das Industrial Object (IO), adressiert Framework-spezifische Implementierungsdetails, die sich bei ACA-Py-Updates ändern. Nach Hamza (2005, S. 1–2) sind IOs konkrete und volatile Repräsentationen der BOs. Diese Volatilität wird bewusst isoliert, sodass Framework-Changes nicht die stabilen EBT- und BO-Module beeinflussen.

Konkret realisiert die Integration-Patching-Schicht die transparente Einbettung durch Monkey-Patching. Kritische ACA-Py-Funktionen werden zur Laufzeit überschrieben, ohne Quellcode zu ändern. Dies ermöglicht PQC-Operationen wie Schlüsselgenerierung, Signatur und Key-Encapsulation in bestehende Workflows. Das Plugin erlaubt auch hybride Modi, bei denen Agenten je nach Plugin-Ladezustand klassische oder quantenresistente Schlüssel erzeugen. Das Monkey-Patching ist auf die volatile IO-Ebene beschränkt, so dass Framework-Updates nicht die stabilen EBT- und BO-Schichten destabilisieren.

Abbildung 16*ACA-Py High Level Applikationsarchitektur mit PQC-Integration*

Anmerkung. Eigene Darstellung basierend auf Abbildung 11.

5.3 Implementierung

Die Implementierung der PQC-Unterstützung auf Application-Ebene folgt einem zweistufigen Ansatz. In der ersten Stufe wird das pqc_didpeer4_fm-Plugin entwickelt, das sich in den ACA-Py Plugin-Mechanismus nach „ACA-Py Plugins - ACA-Py Docs“ (n. d.), „ACA-Py Plugins - ACA-Py Plugins“ (n. d.) und openwallet-foundation (2023, 3. Oktober/2025) einfügt. Das Plugin ist als Python-Paket strukturiert und definiert einen setup-Entry point, der beim Agent-Start aufgerufen wird. Über den PluginContext erhält das Plugin unmittelbaren Zugriff auf zentrale Agent-Komponenten wie die Wallet, das DIDComm-System und die Protocol Registry. Diese Architektur ermöglicht es Plugins, neue Protokolle zu registrieren oder bestehende Funktionen durch Monkey-Patching zu erweitern, ohne den ACA-Py Kern zu modifizieren. Die zweite Stufe integriert das entwickelte Plugin in die containerisierte Deployment-Infrastruktur. Dabei wird das Plugin als Abhängigkeit in dem Dockerfile definiert, sodass es während des Container-Builds installiert wird. Anschließend erfolgt die Konfiguration über docker-compose, indem der ACA-Py Service mit den erforderlichen Umgebungsvariablen und Plugin-Parametern initialisiert wird.

Der Aufbau des PQC-Plugins gliedert sich in drei funktionale Schichten, in die die in Abbildung 17 dargestellten 15 Module eingeteilt sind. Diese Architektur garantiert vollständige PQC-Funktionalität ohne Änderungen am ACA-Py-Quellcode.

Die Kryptografische Abstraktionsschicht bildet die Grundlage und abstrahiert die Komplexität der liboqs-C-Bibliothek durch ein Python-Wrapper-Modul (liboqs_wrapper.py). Sie stellt einheitliche Operationen für ML-DSA-65 (digitale Signaturen) und ML-KEM-768 (Schlüsselencapsulation) bereit und garantiert, dass kryptografisches Schlüsselmaterial als Byte-Sequenzen serialisiert wird.

Die DID-Verarbeitungsschicht orchestriert die vollständige Lebenszyklusbearbeitung von PQC-fähigen did:peer:4-Identifikatoren. Dazu zählen, die Erzeugung und Auflösung (pqc_peer4_creator.py, pqc_peer4_resolver.py), W3C-konforme Multicodecs (pqc_multicodec.py, pqc_multiclientkey.py) und standardisierte DIDComm-Nachrichtenverschlüsselung (pqc_didcomm_v1.py). Diese Schicht verbindet kryptografische Primitive mit Identity-Protokollen und ermöglicht hybrid-sichere Kommunikation zwischen PQC- und klassischen Agenten.

Die Integration-Patching-Schicht implementiert die transparente Einbettung in ACA-Py durch gezieltes Monkey-Patching und Registry-Erweiterungen. Neun spezialisierte Module patchen Wallet-Operationen (askar_pqc_patch.py, wallet_patch.py), Connection-Management (base_manager_patch.py, connection_target_patch.py), Schlüsseltyp-Infrastruktur (key_types.py, key_type_patches.py), Validierungslogik (validator_patch.py),

Multicodec-Registries (`multicodec_patch.py`) und koordinieren deren Installation (`monkey_patches.py`).

Der gesamte Quellcode des `pqc_didpeer4_fm`-Plugins ist in Anhang 4.1 dokumentiert.

Abbildung 17
Aufbau des PQC-Plugins

```

pqc_didpeer4_fm/
  +-- pqc_didpeer4_fm/
    +-- v1_0/
      +-- askar_pqc_patch.py
      +-- base_manager_patch.py
      +-- connection_target_patch.py
      +-- key_type_patches.py
      +-- key_types.py
      +-- liboqs_wrapper.py
      +-- monkey_patches.py
      +-- multicodec_patch.py
      +-- pqc_didcomm_v1.py
      +-- pqc_multicodec.py
      +-- pqc_multikey.py
      +-- pqc_peer4_creator.py
      +-- pqc_peer4_resolver.py
      +-- validator_patch.py
      +-- wallet_patch.py
      +-- __init__.py
    +-- README.md
  +-- setup.py

```

Anmerkung. Eigene Darstellung.

5.3.1 Pluginentwicklung: Kryptografie-Abstraktionsschicht

Die Kryptografie-Abstraktionsschicht bildet die unterste Ebene der PQC-Integration und wird durch das Modul `liboqs_wrapper.py` (Abbildung 18) realisiert. Dieses Modul kapselt die nativen Operationen der C-basierten liboqs-Bibliothek und stellt eine Python-API für die NIST-standardisierten PQC-Algorithmen ML-DSA-65 und ML-KEM-768 bereit.

Abbildung 18*Aufbau des PQC-Plugins - Kryptografie-Abstraktionsschicht*

```

pqc_didpeer4_fm/
  └── pqc_didpeer4_fm/
    └── v1_0/
      ├── askar_pqc_patch.py
      ├── base_manager_patch.py
      ├── connection_target_patch.py
      ├── key_type_patches.py
      ├── key_types.py
      ├── liboqs_wrapper.py
      ├── monkey_patches.py
      ├── multicodec_patch.py
      ├── pqc_didcomm_v1.py
      ├── pqc_multicodec.py
      ├── pqc_multikey.py
      ├── pqc_peer4_creator.py
      ├── pqc_peer4_resolver.py
      ├── validator_patch.py
      └── wallet_patch.py
    └── __init__.py
  └── README.md
└── setup.py

```

Anmerkung. Eigene Darstellung.

Die Implementierung (Listing A-13) definiert eine LibOQSWrapper-Klasse mit sechs Kernmethoden. Die ersten beiden Methoden `generate_ml_dsa_65_keypair()` und `generate_ml_kem_768_keypair()` erzeugen kryptografische Schlüsselpaare. Die Methoden `sign_ml_dsa_65()` und `verify_ml_dsa_65()` implementieren digitale Signaturen, während `encapsulate_ml_kem_768()` und `decapsulate_ml_kem_768()` die Key Encapsulation für sichere Schlüsselvereinbarung realisieren.

Die Wrapper-Architektur abstrahiert die komplexen Foreign-Function-Interface-Aufrufe an die liboqs-C-Bibliothek und stellt sicher, dass Schlüsselmaterial ausschließlich als Byte-Arrays serialisiert wird, was eine Voraussetzung für die Persistierung in der Aries-Askar-Wallet („Entry in Aries_askar::Entry - Rust“, n. d.) und die Kodierung in Multicodec-Formaten (multiformats, 2025) darstellt. Das Singleton-Pattern (`get_liboqs()`) gewährleistet eine einzige globale Instanz zur Vermeidung redundanter Initialisierungen. Diese Abstraktionsschicht ermöglicht es den höheren Modulen (DID-Generierung, DIDComm-Verschlüsselung), PQC-Operationen durchzuführen, ohne direkte Abhängigkeiten zur liboqs-C-API zu haben.

5.3.2 Pluginentwicklung: DID-Verarbeitungsschicht

Die DID-Verarbeitungsschicht (siehe Abbildung 19) orchestriert die Erzeugung, Auflösung und Kodierung von PQC-fähigen did:peer:4-Identifikatoren sowie die DIDComm-Nachrichtenverschlüsselung. Diese Schicht umfasst sechs funktional gekoppelte Module, die gemeinsam eine standardkonforme Integration von PQC in das did:peer:4-Ökosystem realisieren.

Abbildung 19

Aufbau des PQC-Plugins - DID-Verarbeitungsschicht

```
pqc_didpeer4_fm/
└── pqc_didpeer4_fm/
    ├── v1_0/
    │   ├── askar_pqc_patch.py
    │   ├── base_manager_patch.py
    │   ├── connection_target_patch.py
    │   ├── key_type_patches.py
    │   ├── key_types.py
    │   ├── liboqs_wrapper.py
    │   ├── monkey_patches.py
    │   ├── multicodec_patch.py
    │   ├── pqc_didcomm_v1.py
    │   ├── pqc_multicodec.py
    │   ├── pqc_multikey.py
    │   ├── pqc_peer4_creator.py
    │   ├── pqc_peer4_resolver.py
    │   ├── validator_patch.py
    │   └── wallet_patch.py
    └── __init__.py
└── README.md
└── setup.py
```

Anmerkung. Eigene Darstellung.

Das Modul `pqc_peer4_creator.py` (Listing A-14) implementiert die Funktion `create_pqc_peer4_did()`, die aus zwei PQC-Schlüsselpaaren (ML-DSA-65 für authentication/assertionMethod, ML-KEM-768 für keyAgreement) einen did:peer:4-Long-Format-Identifier generiert. Die Schlüssel werden über die Wallet-API erzeugt, in Multikey-Format transformiert und als KeySpec-Objekte in einem did:peer:4-Input-Dokument strukturiert, wobei die Reihenfolge der Schlüssel deren Fragment-IDs determiniert (#key-0 für Signaturen, #key-1 für Verschlüsselung in recipientKeys). Das Gegenstück `pqc_peer4_resolver.py`

(Listing A-15) registriert einen DID-Resolver für die peer-Methode, der did:peer:4-Long-Form-DID in DID-Dokumente auflöst und dabei PQC-Multicodec-Präfixe korrekt dekodiert.

Die Multiformat-Kodierung nach (w3c, 2025, Kap. 5.6) wird durch drei Module realisiert. Das Modul pqc_multicodec.py (Listing A-16) definiert eine Multicodec-Registry mit provisorischen Präfixen (ML-DSA-65: 0xd065, ML-KEM-768: 0xe018) und stellt Wrapper-Funktionen (wrap_pqc(), unwrap_pqc()) für Präfix-Operationen bereit. Das Modul pqc_multikey.py (Listing A-17) transformiert Schlüsselinformationen in das Multikey-Format durch Verkettung von Multicodec-Präfix und Schlüsselmaterial sowie Base58-Kodierung mit Multibase-Präfix „z“ (Base58btc), wodurch Multikeys wie „z6MNxxx...“ (ML-DSA-65) oder „z6MK768xxx...“ (ML-KEM-768) entstehen. Das Modul pqc_didcomm_v1.py (Listing A-18) erweitert die DIDComm-v1-Envelope-Verarbeitung um PQC-Unterstützung. Hier detektieren die beiden Methoden pack_message_pqc() und unpack_message_pqc() automatisch anhand der unterschiedlichen Schlüssellängen ob PQC- oder klassische Kryptografie verwendet werden muss, und generieren JWE-Envelopes mit angepassten Algorithmus-Headern. Die Content Encryption erfolgt weiterhin mit XChaCha20-Poly1305, während der Content Encryption Key mittels ML-KEM-768 Key Encapsulation für jeden Empfänger verschlüsselt wird. Diese Schicht ermöglicht eine hybride Betriebsweise, bei der PQC- und klassische Agenten koexistieren können, solange jeweils homogene Verschlüsselungsmodi verwendet werden.

5.3.3 Pluginentwicklung: Integration-Patching-Schicht

Die Integration-Patching-Schicht (Abbildung 20) implementiert die transparente Einbettung der PQC-Funktionalität in den ACA-Py-Kern durch gezieltes Monkey-Patching kritischer Funktionen und Erweiterung globaler Registries. Diese Schicht umfasst neun Module, die gemeinsam eine vollständig transparente PQC-Integration ohne Modifikation des ACA-Py-Quellcodes ermöglichen, sodass existierende Workflows und API-Aufrufe unverändert funktionsfähig bleiben.

Abbildung 20
Aufbau des PQC-Plugins - Integration-Patching-Schicht

```
pqc_didpeer4_fm/
  pqc_didpeer4_fm/
    v1_0/
      askar_pqc_patch.py
      base_manager_patch.py
      connection_target_patch.py
      key_type_patches.py
      key_types.py
      liboqs_wrapper.py
      monkey_patches.py
      multicodec_patch.py
      pqc_didcomm_v1.py
      pqc_multicodec.py
      pqc_multikey.py
      pqc_peer4_creator.py
      pqc_peer4_resolver.py
      validator_patch.py
      wallet_patch.py
      __init__.py
    README.md
  setup.py
```

Anmerkung. Eigene Darstellung.

Das zentrale Orchestrierungsmodul `monkey_patches.py` (Listing A-19) koordiniert die Installation aller Patches durch die Funktion `apply_all_patches()`, die beim Plugin-Setup aufgerufen wird. Dieses Modul überschreibt Methoden der Klasse `BaseConnectionManager` (`create_did_peer_4()`, `_extract_key_material_in_base58_format()`, `long_did_peer_4_to_short()`) und delegiert deren Implementierung an spezialisierte Patch-Module, wobei die ursprünglichen Methoden als Fallback-Referenzen gespeichert werden. Das Modul `base_manager_patch.py` (Listing A-20) stellt die PQC-Implementierungen dieser `BaseConnectionManager`-Methoden bereit. Die Methode `create_did_peer_4_pqc_complete()` generiert `did:peer:4-DID` mit ML-DSA-65- und ML-KEM-768-Schlüsseln anstelle klassischer ED25519/X25519-Schlüssel, die Methode `_extract_key_material_in_base58_format_pqc()` extrahiert PQC-Schlüsselmaterial aus DID-Dokumenten unter Berücksichtigung der größeren Schlüssellängen, und die Methode `record_keys_for_resolvable_did_pqc()` persistiert beide PQC-Schlüssel (Signatur- und Verschlüsselungsschlüssel) in der Wallet-Datenbank.

Die Wallet-Integration erfolgt durch drei Module. Das Modul `askar_pqc_patch.py` (Listing A-21) patcht die Aries-Askar-Funktionen `create_keypair()` zur Unterstützung von PQC-Schlüsselgenerierung mittels `liboqs` sowie `pack_message()` und `unpack_message()` zur Integration der PQC-DIDComm-v1-Implementierung aus `pqc_didcomm_v1.py`. Das Modul `wallet_patch.py` (Listing A-22) erweitert die Methode `get_local_did_for_verkey()` der `AskarWallet`-Klasse, um ML-KEM-768-Verkeys korrekt in der Datenbank zu lokalisieren. Dies stellt eine kritische Anpassung dar, da klassische Verkey-Lookups, wie in der Originalmethode `get_local_did_for_verkey()` (`openwallet-foundation`, n. d. k) demonstriert, nur für 32-Byte-ED25519-Schlüsselelemente ausgelegt sind. Das Modul `connection_target_patch.py` (Listing A-23) passt das Marshmallow-Schema der `ConnectionTarget`-Klasse an, indem die Validierungsregeln für `recipient_keys` PQC-konforme Schlüssellängen akzeptieren.

Die Erweiterung der Schlüsseltyp-Infrastruktur erfolgt durch zwei Module. Das Modul `key_types.py` (Listing A-25) definiert neue KeyType-Konstanten (ML_DSA_65, ML_KEM_768) mit Metadaten wie NIST-FIPS-Referenzen, Schlüssellängen und Multicodec-Präfixen. Das Modul `key_type_patches.py` (Listing A-26) registriert diese KeyTypes in der globalen ACA-Py-Registry durch `register_pqc_key_types()`, erweitert die Admin-API-Schemata (`patch_api_key_type_schemas()`) zur Akzeptanz von PQC-KeyType-Strings in JSON-Requests, und patcht Algorithmus-Mappings (`patch_alg_mappings_for_pqc()`) für JWS/JWE-Header-Generierung. Das Modul `multicodec_patch.py` (Listing A-27) erweitert die globale `SupportedCodecs`-Enumeration durch dynamisches Hinzufügen von ML-DSA-65- und ML-KEM-768-Multicodec-Einträgen, so dass Multicodec-Dekodierungsfunktionen aus multiformats-Bibliotheken PQC-Präfixe verarbeiten können.

Das Modul `validator_patch.py` (Listing A-24) patcht die `JWSHeaderKid`-Validierungsklasse, die standardmäßig nur klassische DID-Formate (`did:key`, `did:sov`) in JWS-Header-`kid`-Feldern akzeptiert, um `did:peer:4`-Identifier zu unterstützen. Dies ist eine Voraussetzung für ML-DSA-65-signierte DID-Exchange-AttachDecorators.

Diese neun Module bilden gemeinsam eine Patch-Architektur, die durch sequenzielle Installation beim Plugin-Setup (orchestriert in `__init__.py`) eine vollständige PQC-Funktionalität in ACA-Py injiziert, ohne dass Änderungen an Controllern, Admin-API-Endpunkten oder externen Business-Logic-Schichten erforderlich sind.

5.3.4 Dockerfile-Modifikation

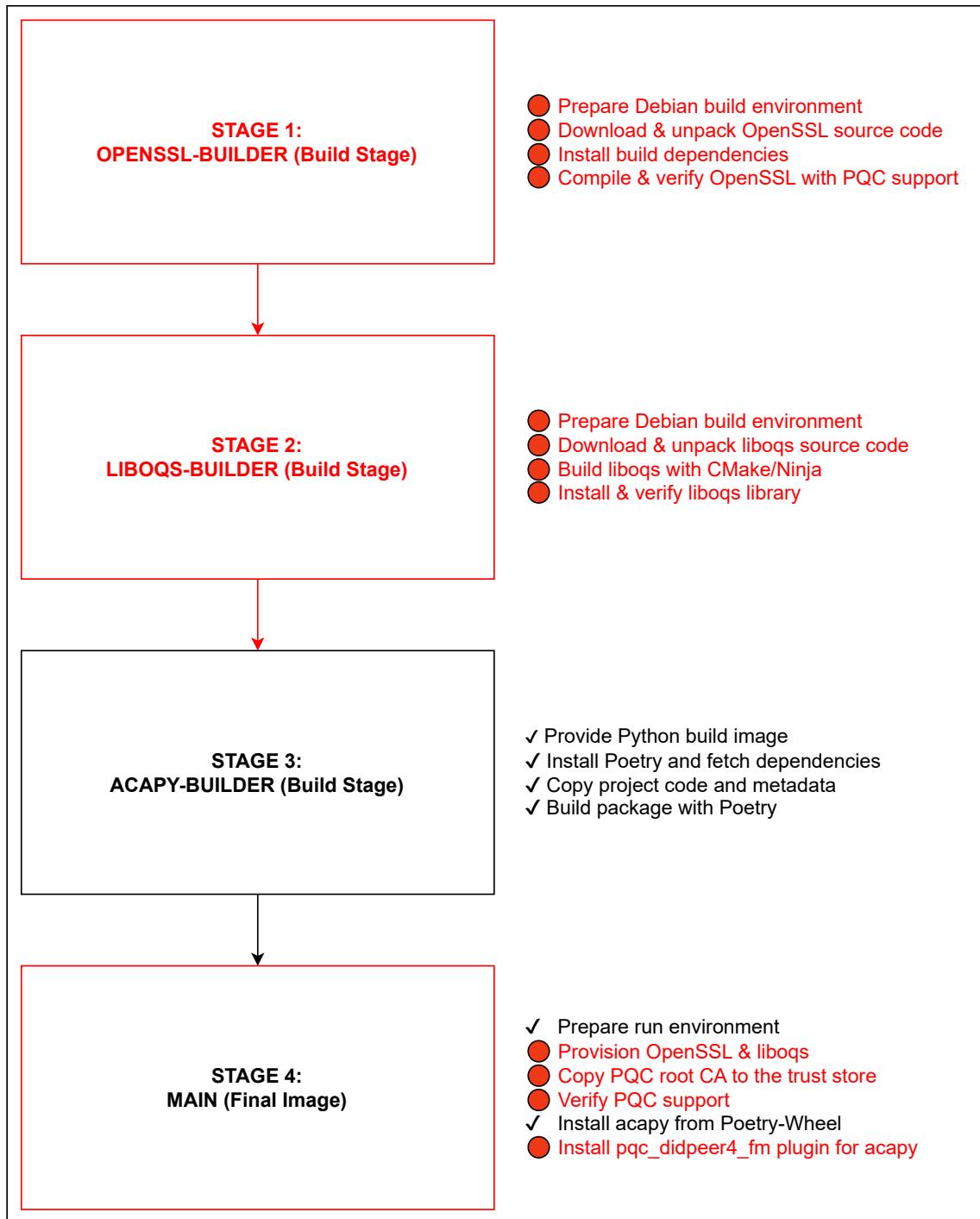
In der zweiten Iteration wurde das ACA-Py Docker-Base-Image (Listing A-7) aus der ersten Iteration (Kapitel 4.5.5) auf einen vier-stufigen Multi-Stage-Build erweitert (Listing A-30 visualisiert dargestellt in Abbildung 21).

Die ersten beiden Stufen widmen sich dem kryptografischen Fundament. Da aktuelle Distributionen die benötigten Verfahren noch nicht beinhalten, werden die Bibliotheken hier direkt aus dem Quellcode kompiliert. In der ersten Stufe wird OpenSSL in der Version 3.5.4 kompiliert. Dabei werden gezielt Parameter gesetzt, die das Modul auf FIPS-Konformität vorbereiten, was für den späteren Einsatz in KRITIS relevant ist. Parallel dazu wird in der zweiten Stufe die liboqs-Bibliothek erstellt. Sie fungiert als Backend für die PQC-Algorithmen und wird so konfiguriert, dass sie als dynamische Bibliothek vorliegt und später nahtlos von der Python-Umgebung eingebunden werden kann.

Die dritte Stufe konzentriert sich anschließend isoliert auf die Anwendungsebene. Hier wird der ACA-Py-Agent mithilfe des Tools Poetry („Poetry - Python Dependency Management and Packaging Made Easy“, n. d.) in ein installierbares Python-Paket verpackt. Diese methodische Trennung setzt die wissenschaftliche Forderung nach einer Minimierung des Trusted Computing Base nach Jarkas et al. (2025, S. 27) um, indem reine Entwicklungswerkzeuge nicht in das finale Image übernommen werden. Somit wird der Speicherbedarf reduziert und die Sicherheit des Artefakts durch eine verringerte Angriffsfläche erhöht.

In der finalen Stufe fließen alle vorbereiteten Komponenten zusammen. Der technisch wichtigste Schritt ist hierbei der Austausch der Standard-Kryptografie. Durch das gezielte Überschreiben von Systemverknüpfungen wird erzwungen, dass sowohl das Betriebssystem als auch die Python-Laufzeitumgebung automatisch auf die zuvor kompilierte, PQPQCC-fähige OpenSSL-Version zugreifen. Zusätzlich wird das eigene Root-Zertifikat in den Vertrauensspeicher importiert, damit sichere TLS-Verbindungen korrekt validiert werden können. Den Abschluss bildet die Installation des spezifischen Plugins, das die erweiterte Logik für die Verarbeitung der dezentralen Identifikatoren bereitstellt.

Abbildung 21
ACA-Py Multi-Stage Build Dockerfile mit PQC-Integration



Anmerkung. Eigene Darstellung basierend auf Listing A-30.

5.3.5 Deployment in docker-compose.yml

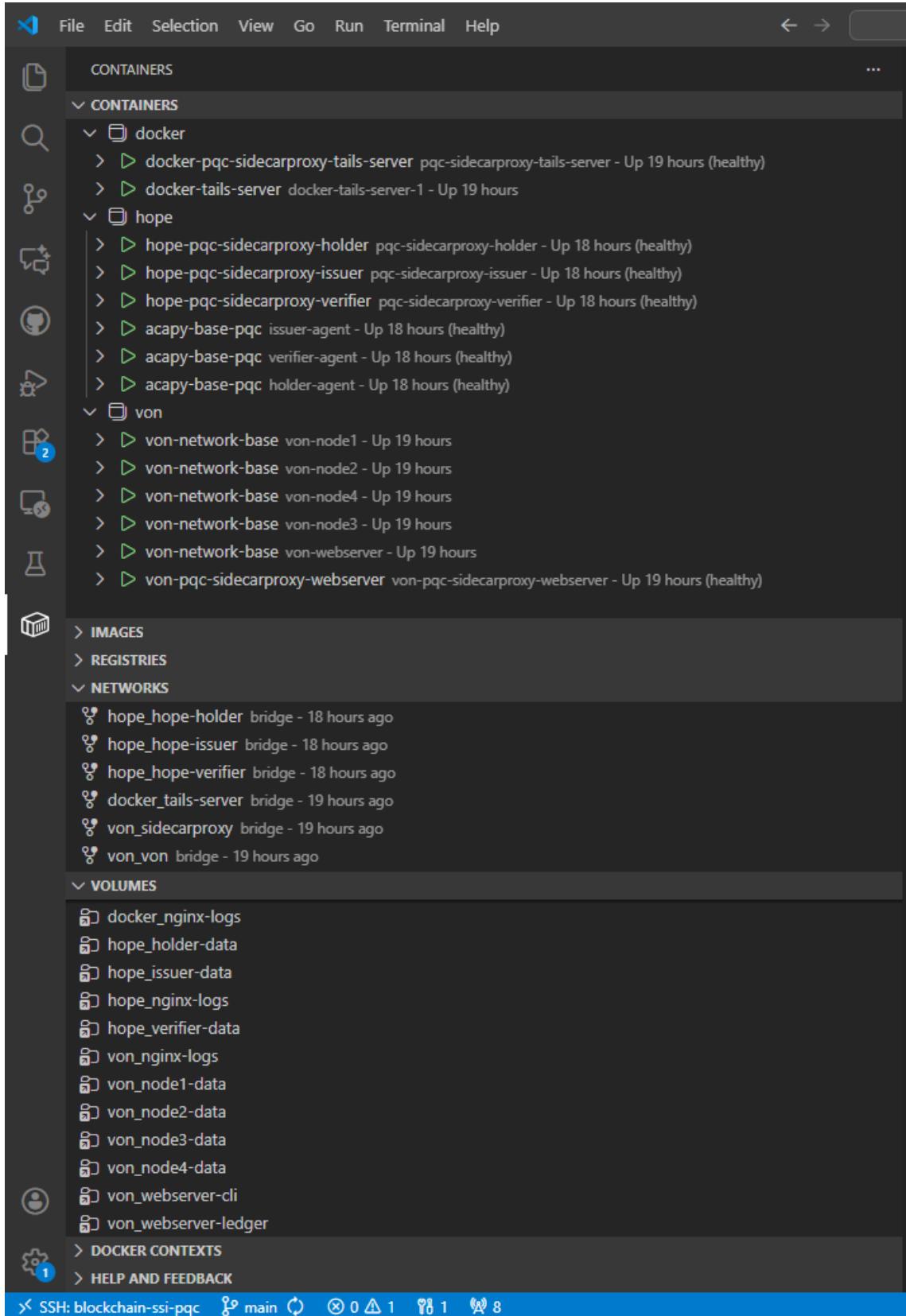
Das Deployment der PQC-fähigen SSI-Agenten erfolgt innerhalb der in der ersten Iteration (Kapitel 4.5.6) entwickelten docker-compose.yml-Orchestrierung, deren Evolution vom klassischen Setup (erste Iteration, Listing A-8) zur PQC-Integration (zweite Iteration, Listing A-31) zwei zentrale Anpassungen umfasst. Während die Konfiguration der ersten Iteration noch das klassische ACA-Py-Base-Image ohne PQC-Unterstützung und Plugin-Aktivierung verwendet, wurde in der zweiten Iteration im Rahmen der ersten Anpassung die docker-compose.yml so modifiziert, dass alle drei Agent-Services (issuer, holder, verifier) das neue acapy-base-pqc-Image (basierend auf dem Dockerfile aus Kapitel 5.3.4) nutzen in welchem das pqc_didpeer4_fm-Plugin enthalten ist. Diese Änderung kann durch den Vergleich von Abbildung 14 mit Abbildung 22 nachvollzogen werden. Die zweite Anpassung erweitert die command-Direktive aller drei Agenten um den Parameter „–plugin pqc_didpeer4_fm“, der beim Agent start das pqc_didpeer4_fm-Plugin lädt.

5.4 Formative Evaluation

In Übereinstimmung mit dem in Kapitel 3.3.4 entworfenen Evaluationsdesign setzt dieser Abschnitt die zweite definierte Evaluationsepisode (Tabelle 3) um. Charakterisiert als formative und artifizielle Untersuchung, liegt der Fokus dieser Phase exklusiv auf der Validierung der Plugin-Integrationsarchitektur sowie der korrekten Funktionalität der kryptographischen Abstraktionsschichten während der Agent-Laufzeit. Ziel ist es, gemäß den Prinzipien von Venable, Pries-Heje und Baskerville (2016, S. 82–83) Designfehler in der Monkey-Patching-Strategie und der DID-Verarbeitungskette frühzeitig zu identifizieren und die Integrationsfähigkeit der PQC in die bestehende ACA-Py-Infrastruktur nachzuweisen.

Zentrale Evaluationsziele dieser Phase sind die Korrektheit des Plugin-Ladevorgangs bei der Agenten-Initialisierung sowie die Validierung der Post-Quantum-Funktionalität im Kontext von did:peer:4-basierter Identifiziererstellung. Methodisch erfolgt die Evaluation durch kontrollierte Integrationstests der Plugin-Architektur, DIDComm-Verarbeitung und Validierung der PQC-Signaturen in Form von White-Box Tests nach Myers, Sandler und Badgett (2012, S. 10).

Abbildung 22
Docker-Compose-Übersicht der zweiten Iteration



Anmerkung. Eigene Darstellung.

5.4.1 Validierung des Plugin-Ladevorgangs bei Agent-Start

Die erste formative Evaluationsmaßnahme besteht in der Validierung des korrekten Plugin-Ladevorgangs beim Start eines ACA-Py-Agenten. Dieser Test dient der Sicherstellung, dass die PQC-Integration transparent und ohne Beeinträchtigung der Standard-ACA-Py-Funktionalität erfolgt.

Listing A-12 zeigt den Boot-Prozess eines Standard-ACA-Py-Agenten ohne PQC-Plugin. Nach der Registrierung der Default- und Askar-Plugins wird direkt mit der Ledger-Konfiguration und Wallet-Initialisierung fortgefahren. Im Vergleich dazu zeigt Listing A-32 den erweiterten Boot-Prozess mit geladenem pqc_didpeer4_fm-Plugin. Zwischen der Askar-Plugin-Registrierung und der Ledger-Konfiguration erfolgt nun die Plugin-Initialisierung mit mehreren charakteristischen Schritten.

1. **Askar-Patching:** Die `_create_keypair`-Funktion wird durch eine PQC-fähige Variante ersetzt, die ML-DSA-65 und ML-KEM-768 unterstützt. Zusätzlich werden Session-Methoden (`insert_key`, `fetch_key`, `update_key`) und `AskarWallet.assign_kid_to_key()` gepatcht.
2. **KeyType-Registry-Erweiterung:** Die neuen Schlüsseltypen ML-DSA-65 und ML-KEM-768 werden in der ACA-Py KeyTypes-Registry registriert und die API-Schemas zur Laufzeit erweitert.
3. **did:peer:4-Erweiterung:** Die unterstützten Schlüsseltypen für `did:peer:4` werden von `['ed25519', 'x25519']` auf `['ed25519', 'x25519', 'ML-DSA-65', 'ML-KEM-768']` erweitert.
4. **Multicodec-Patching:** Die `SupportedCodecs`-Klasse wird für PQC-Multicodec-Präfixe erweitert (ML-DSA-65: `0xd065`, ML-KEM-768: `0xe018`).
5. **DIDComm-Patching:** `AskarWallet.pack_message()` und `unpack_message()` werden für ML-KEM-768-basierte Verschlüsselung angepasst. Die `AttachDecorator`-Klasse wird für ML-DSA-65-JWS-Signaturen erweitert.
6. **Monkey-Patches:** Die `BaseConnectionManager`-Methoden (`create_did_peer_4`, `record_keys_for_resolvable_did`, etc.) werden durch PQC-fähige Varianten ersetzt.

5.4.2 Validierung der Pluginfunktionalität

Die zweite formative Evaluationsmaßnahme validiert die Kernfunktionalität des Plugins, welche die transparente Erstellung von PQC-fähigen `did:peer:4-DID` während des Out-of-Band-Invitation-Prozesses darstellt.

Listing 1 zeigt die initiale Wallet-Abfrage eines frisch gestarteten Issuer-Agenten. Das leere results-Array bestätigt, dass noch keine DID im Wallet vorhanden sind.

Listing 1

Zweite Iteration - Wallet DID Abfrage vor Out-of-Band Invitation

```

1 ferris@blockchain-ssi-pqc:~$ curl -X GET https://host.docker.internal:8021/wallet/did
   | jq
2   % Total      % Received % Xferd  Average Speed   Time     Time     Time  Current
3                               Dload  Upload   Total  Spent   Left  Speed
4 100    15  100    15     0      0   1083      0 --:--:-- --:--:-- --:--:--  1153
5 {
6   "results": []
7 }
```

Anschließend wird mittels „POST /out-of-band/create-invitation“ mit dem Parameter „use_did_method: 'did:peer:4'“ eine Einladung erstellt (Listing 32). Die API-Response enthält eine vollständige did:peer:4-Langform-DID im services-Array der Invitation, erkennbar am charakteristischen Format „did:peer:4zQm...:z25g...“.

Listing 2

Zweite Iteration - Out-of-Band Invitation

```

1 ferris@blockchain-ssi-pqc:~$ curl -X POST https://host.docker.internal:8021/out-of-
2   band/create-invitation -H "Content-Type: application/json" -d '{
3     "handshake_protocols": ["https://didcomm.org/didexchange/1.1"],
4     "use_did_method": "did:peer:4",
5     "my_label": "Issuer Test"
6   }' | jq
7   % Total      % Received % Xferd  Average Speed   Time     Time     Time  Current
8                               Dload  Upload   Total  Spent   Left  Speed
9 100 16198  100 16051  100    147  91160      834 --:--:-- --:--:-- --:--:--  91514
10 {
11   "state": "initial",
12   "trace": false,
13   "invi_msg_id": "89e9cc87-318f-49aa-a61a-fc805706cd8d",
14   "oob_id": "70998122-5a5b-4020-8b5f-ae5884af20b3",
15   "invitation": {
16     "@type": "https://didcomm.org/out-of-band/1.1/invitation",
17     "@id": "89e9cc87-318f-49aa-a61a-fc805706cd8d",
18     "label": "Issuer Test",
19     "handshake_protocols": [
20       "https://didcomm.org/didexchange/1.1"
21     ],
22     "services": [
23       "did:peer:4zQmYFdntsqaizCzU9PMf4dVshmxyTu5yk3NnkA28VjHqaySm:
24         z25gYmQoBS9XWQbLxdKXKizWUz5MxCWwLc..."
25     ]
26   },
27   "invitation_url": "https://host.docker.internal:8020?oob=eyJAdHlwZSI6ICJodHR..."
```

Die entscheidende Validierung erfolgt in Listing 3 durch eine erneute Wallet-Abfrage nach der Invitation-Erstellung. Die Response zeigt nun die automatisch generierte PQC-DID mit folgenden charakteristischen Merkmalen:

- **Dual-Key-Struktur:** Das key_type-Feld weist den Wert „ml-dsa-65“ auf, während die Metadata zusätzlich kem_verkey (ML-KEM-768) enthält. Dies bestätigt die erfolgreiche Implementierung der Hybrid-Kryptografie mit getrennten Schlüsseln für digitale Signaturen und Schlüsselvereinbarung.
- **PQC-Metadata:** Die Metadaten enthalten explizite Marker (pqc_enabled: true, signature_algorithm: "ml-dsa-65", key_agreement_algorithm: "ml-kem-768"), die eine eindeutige Identifikation PQC-fähiger DID zur Laufzeit ermöglichen.
- **Key Identifier:** Das kem_key_kid-Feld referenziert den KEM-Schlüssel über den DID-URL-Fragment-Identifier #key-1, was der did:peer:4-Spezifikation entspricht, bei der Verification-Methods sequenziell nummeriert werden, sodass #key-0 für Authentication und #key-1 für Key Agreement genutzt wird (decentralized-identity, n. d. d.).

Listing 3

Zweite Iteration - Wallet DID Abfrage nach Out-of-Band Invitation

```

1 ferris@blockchain-ssi-pqc:~$ curl -X GET https://host.docker.internal:8021/wallet/did
2   | jq
3
4   % Total      % Received % Xferd  Average Speed   Time     Time     Time  Current
5   |          |          |          |          |          |          |          |          |
6   |          |          |          |          |          |          |          |          |
7   |          |          |          |          |          |          |          |          |
8   |          |          |          |          |          |          |          |          |
9   |          |          |          |          |          |          |          |          |
10  |          |          |          |          |          |          |          |          |
11  |          |          |          |          |          |          |          |          |
12  |          |          |          |          |          |          |          |          |
13  |          |          |          |          |          |          |          |          |
14  |          |          |          |          |          |          |          |          |
15  |          |          |          |          |          |          |          |          |
16  |          |          |          |          |          |          |          |          |
17  |          |          |          |          |          |          |          |          |
18  |          |          |          |          |          |          |          |          |
19  |          |          |          |          |          |          |          |          |
20  |          |          |          |          |          |          |          |          |
21  |          |          |          |          |          |          |          |          |
22  |          |          |          |          |          |          |          |          |
23  |          |          |          |          |          |          |          |          |
24  |          |          |          |          |          |          |          |          |
25 }
```

5.5 Finales Artefakt

Das finale Artefakt der zweiten Iteration repräsentiert einen durchgängig quantensicheren SSI-Prototypen mit nativer PQC-Integration auf der Anwendungsebene. Die Architektur vereint die in der ersten Iteration etablierte TLS mittels nginx-PQC-Sidecar-Proxies mit einer tiefgreifenden Applikationsschicht-Integration durch das entwickelte PQC-Plugin „`pqc_didpeer4_fm`“. Diese duale Strategie gewährleistet nicht nur die Absicherung von Data-In-Motion, sondern auch die langfristige Authentizität und Integrität von Data-At-Rest.

Die Kernkomponente bildet das ACA-Py-Plugin mit dreischichtiger Architektur nach dem in Kapitel 5.2.2 beschriebenen Software Stability Model. Zur Realisierung langfristiger Wartbarkeit folgt das Plugin intern dem Ansatz der Separation of Concerns for Evolving Systems nach Hamza (2005, S. 1), der Systeme entlang der Stabilitätsdimension dekomponiert, um die Notwendigkeit zur Reseparation bei System-Evolution zu minimieren.

Die stabile Kern-Ebene (EBT) kapselt native liboqs 0.14.0-Operationen als abstrakte Kryptografie-Primitiva für ML-DSA-65 und ML-KEM-768. Diese Abstraktionsschicht ist unabhängig vom konkreten Algorithmus und bleibt stabil. Wenn zukünftig ML-DSA-87 verwendet wird, ändert sich nur die Implementierung, nicht die Schnittstelle. Nach Hamza sind EBTs die Basis für Stabilität, da Systeme ohne EBTs bei Änderungen resepariert werden müssen (Hamza, 2005, S. 1–2).

Die semi-stabile Geschäfts-Objekt-Schicht (BO) orchestriert die Generierung, Auflösung und Kodierung von PQC-fähigen did:peer:4-Identifikatoren sowie die DIDComm-Nachrichtenverschlüsselung. BOs sind nach Hamza intern adaptierbar durch sogenannte hooks, aber extern stabil (Hamza, 2005, S. 2). Die BO-Implementierung umfasst W3C-konforme DID-Verarbeitung, Multicodec-Identifikatoren und Key-Type-Registry-Erweiterungen. Bei Migration zu ML-KEM-1024 können neue Multicodec-Präfixe hinzugefügt werden, ohne die DIDComm-Schnittstellen zu brechen.

Die volatile Implementierungs-Ebene (IO) realisiert transparentes Monkey-Patching kritischer ACA-Py-Kernfunktionen (Wallet, Key-Type-Registry, DIDComm-Verschlüsselung) ohne Modifikation des Framework-Quellcodes. Dies ermöglicht eine hybride Betriebsweise, bei der Agenten je nach Plugin-Ladezustand klassische oder quantenresistente Schlüssel erzeugen. Nach Hamza sollten Änderungen an IO-Modulen, wie beispielsweise der Austausch von Speichersystemen, nicht die stabileren Schichten EBT und BO beeinflussen (Hamza, 2005, S. 4).

Hinsichtlich der in Kapitel 5.1 definierten Designziele erfüllt das finale Artefakt sämtliche Anforderungen der zweiten Iteration vollständig. Das Designziel zu FF1 (Systemarchitektur & Compliance) wird durch die native Unterstützung quantenresistenter Signaturen in DID-Dokumenten und VC adressiert. Das Designziel zu FF2 (Algorithmenauswahl & Sicherheitsbewertung) manifestiert sich in der erfolgreichen Integration von ML-DSA-65 für digitale Signaturen und ML-KEM-768 für Key-Encapsulation innerhalb der did:peer:4-Strukturen. Das Designziel zu FF3 (Kryptografische Agilität) wird durch die dreischichtige Separation nach (Hamza, 2005) realisiert. Neue Algorithmen können durch BO-Erweiterung integriert werden, ohne EBT zu ändern (Extensibility). Degradierte Algorithmen können durch Konfigurationsänderung deaktiviert werden, ohne dass die Kern-Architektur modifiziert wird (Reversibility). Die nahtlose Koexistenz und der hybride Betrieb zwischen klassischen Verfahren und PQC-Verfahren erfolgen ohne Beeinträchtigung der Interoperabilität, da die abstrahierte Kryptografie-Schicht die unterschiedlichen Algorithmen transparent verwaltet. Das Plugin kann vollständig deaktiviert werden, ohne dass die zugrunde liegende ACA-Py-Architektur destabilisiert wird.

Die formative Evaluation in Kapitel 5.4 validiert die funktionale Korrektheit durch die erfolgreiche Plugin-Registrierung und Boot-Logging bei der Agenten-Initialisierung, die korrekte Generierung von did:peer:4-Long-Form-DID mit ML-DSA-65-Signaturmaterial, die erfolgreiche Out-of-Band-Invitation mit anschließender Wallet-Persistierung von PQC-Metadaten und die fehlerfreie Integration der Monkey-Patching-Funktionalität in alle vier klassischen Applikationsschichten.

6 Summative Evaluation

Die summative Evaluation validiert das finale Artefakt gegen die in Kapitel 3.3 definierten Evaluationsziele sowie die funktionalen und Compliance-Anforderungen aus Kapitel 4.2. Im Gegensatz zu den formativen Evaluationsepisoden (Kapitel 4.6 und Kapitel 5.4), die modulare Einzelkomponenten prüften, erfolgt die summative Evaluation als systematische Requirement-Tracing am integrierten Gesamtsystem. Das Ziel besteht im rigorosen Nachweis der Efficacy und Fidelity, um die technische Machbarkeit und regulatorische Konformität des PQC-basierten SSI-Prototypen für KRITIS zu demonstrieren.

Die Evaluierung wird anhand eines domänenrealistischen KRITIS-Szenarios durchgeführt (Anhang 5.1), das einen Notfall-Wartungszertifizierungsprozess in einem Stromversorgungsnetz abbildet und alle sechs funktionalen Anforderungen integriert. Die detaillierte technische Dokumentation ist in Anhang 5 aufgeführt und referenziert Jupyter-Notebook-Cells (Listing A-33 bis Listing A-83).

Bevor die eigentliche Validierung der funktionalen Anforderungen beginnt, ist zunächst die Initialisierung des Artefakts erforderlich. Das technische Setup deklariert zentrale Variablen, Basis-URLs der Admin-APIs sowie Hilfsfunktionen zur HTTP-Kommunikation (Listing A-33). Auf dieser Grundlage wird die Infrastruktur überprüft und die erfolgreiche Ledger-Initialisierung durch Abruf der Genesis-Transaktionen und Validator-Node-Registrierungen nachgewiesen, womit alle benötigten Kernkomponenten (Indy-Ledger, Validator-Nodes, Tails-Server) in konsistentem Zustand vorliegen (Listing A-36).

Die domänenrelevante Identitätsinfrastruktur für den KRITIS-Issuer wird vorbereitet durch lokale Erzeugung einer neuen Indy-DID als kryptographische Identität des Energienetzbetreibers (Listing A-38). Diese DID wird via NYM-Transaktion als ENDORSER auf dem Ledger registriert, womit der Issuer als vertrauenswürdiger Teilnehmer verankert wird (Listing A-40). Die Wallet-Ansicht bestätigt das Ed25519-Schlüsselmaterial und Status „posted=true“, womit die On-Ledger-Publikation nachgewiesen wird (Listing A-42).

Die fachlichen Grundlagen für die Zertifikatsausstellung werden geschaffen durch ein spezifisches Schema für das KRITIS-Notfall-Wartungszertifikat mit neun Attributen (Listing A-44). Eine Issuer-spezifische Credential Definition mit zugehöriger Revocation Registry wird erzeugt, die kryptographische Parameter und Revokationsmechanismen festlegt (Listing A-46). Mit Abschluss dieser Initialisierungsschritte liegt ein vollständig instanziertes Artefakt auf reproduzierbarem, domänenkonsistentem Fundament vor.

6.1 Validierung der funktionalen Anforderungen

6.1.1 Issuer Discovery

Die funktionale Anforderung FR1 fordert, dass das System die Auffindbarkeit von publizierten Credential-Schemata des Issuers digitaler Identitätsnachweise ermöglichen muss. Die Erfüllung dieser Anforderung an das finale Artefakt wird anhand eines dreiphasigen, Ledger-basierten Discovery-Mechanismus demonstriert (Listing A-47 und Listing A-48).

Phase 1 extrahiert alle TRUST_ANCHOR-Identitäten (Role '101') aus NYM-Transaktionen des Domain Ledgers, wobei im KRITIS-Szenario der Issuer „Energienetzbetreiber“ mit DID „9pbXiFBZZGwXKp61HQBz3J“ identifiziert wird (Listing A-48).

Phase 2 verifiziert sechs kryptographische Eigenschaften (DID-Identifier, Ed25519-Verkey, TRUST_ANCHOR-Role, Endorser, On-Ledger-Aktivitäten, Registrierungszeitpunkt) mittels der Funktion `verify_issuer_identity()`, wobei für den identifizierten Issuer alle Eigenschaften erfolgreich validiert werden (Listing A-48).

Phase 3 filtert SCHEMA-Transaktionen nach dem Schema-Namen „kritis_emergency_maintenance_cert“, extrahiert den Issuer-DID aus dem Schema-Identifier-Format „`<issuer_did>:2:<schema_name>:<version>`“ und führt eine Cross-Referenzierung mit den TRUST_ANCHOR-Identitäten durch (Listing A-48).

6.1.2 Connection Creation

Die funktionale Anforderung FR2 verlangt die Etablierung von Verbindungen zwischen SSI-Akteuren. Dies wird mittels eines dreiphasigen Out-of-Band-Protokoll-Workflows mit did:peer:4-basierter PQC demonstriert.

Phase 1 implementiert einen Pre-Check existierender Connections via „GET /connections“ auf beiden Agenten, um redundante Connection-Erstellungen zu vermeiden. Im KRITIS-Szenario werden keine existierenden Connections gefunden, wodurch eine neue Etablierung ausgelöst wird (Listing A-50).

Phase 2 realisiert die Connection-Etablierung mittels Aries RFC 0434 Out-of-Band Protocol (decentralized-identity, n. d. b). Der Inviter erstellt eine Invitation mit „use_did_method: 'did:peer:4'“ und erhält eine `invitation_msg_id` als eindeutigen Identifier (Listing A-50). Der Invitee akzeptiert die Invitation, wodurch das DIDEExchange-Protokoll did:peer:4-DID

mit ML-DSA-65- sowie ML-KEM-768-Schlüsselmaterial generiert (Listing A-52 und Listing A-56). Die resultierenden DID-Metadaten bestätigen „pqc_enabled: true“, „signature_algorithm: ml-dsa-65“ und „key_agreement_algorithm: ml-kem-768“.

Phase 3 validiert die Connection-Konsistenz durch Vergleich der invitation_msg_id, komplementärer their_role-Werte (inviter/invitee) und beidseitigen State active (Listing A-50). Die Connection-Übersicht (Listing A-54) zeigt zwei aktive Connection-Paare (Issuer ↔ Holder und Holder ↔ Verifier), womit die vollständige Konnektivität des SSI-Dreiecks nachgewiesen wird.

6.1.3 Credential Creation

Die funktionale Anforderung FR3 fordert, dass das System Funktionalität zur Erstellung und Ausstellung digitaler Credentials bereitstellen muss. Die Erfüllung wird anhand eines mehrstufigen Credential-Issuance-Workflows mit Revocation-Registry-Integration demonstriert.

Der Issuer initiiert die Credential-Ausstellung durch Versenden eines Credential Offers mit einer Credential Preview, welche neun KRITIS-spezifische Attribute enthält. Die Ausstellung erfolgt dabei über die in FR2 etablierte connection_id und die in FR1 identifizierte cred_def_id (Listing A-58). Der Holder akzeptiert das Offer automatisch, wodurch das Aries RFC 0453 Issue Credential v2.0 Protocol (decentralized-identity, n. d. c) den vollständigen State-Machine-Durchlauf (offer-sent → request-sent → credential-issued → done) ausführt und das Credential im Holder Wallet persistiert (Listing A-58).

Die Revocation-Registry-Integration extrahiert zwei kritische Identifier aus der Issuer-Exchange-Response. Die Revocation Registry ID identifiziert die auf dem Indy Ledger publizierte Revocation Registry (Type 113), während die Credential Revocation ID „1“ die Position im Revocation-Accumulator spezifiziert (Listing A-58). Die Holder-Credentials-Übersicht (Listing A-60) zeigt das vollständig ausgestellte KRITIS-Notfall-Wartungszertifikat mit allen Attributen, dem Schema-Identifier aus FR1 und dem initialen Revoked-Status false, womit die Gültigkeit bestätigt wird.

6.1.4 Verification with Credentials

Die funktionale Anforderung FR4 fordert, dass das System einen Verifikationsprozess zwischen Identity Holder, Verifier und blockchain-basierter VDR durch Validierung eines Identitätsnachweises ermöglichen muss. Die Erfüllung wird anhand eines vierstufigen Privacy-Preserving-Verification-Workflows mit ZKP, Revocation-Detection und Zeitgültigkeitsprüfung demonstriert.

Der Verifier initiiert den Verifikationsprozess durch Versenden eines Proof Requests mit einer Indy-Proof-Request-Struktur, die fünf offengelegte Attribute (cert_type, facility_type, epoch_valid_from, epoch_valid_until, role) und ein Zero-Knowledge-Predicate (security_clearance_level >= 2) fordert, während drei Identitätsattribute (first_name, name, organisation) durch Selective Disclosure geschützt bleiben (Listing A-61 und A-62). Alle Attribute enthalten eine non_revoked-Constraint mit Zeitintervall, die eine Ledger-basierte Echtzeit-Revocation-Prüfung gegen die Revocation Registry erzwingt.

Der Holder empfängt den Proof Request und prüft verfügbare Credentials für die Präsentation (Listing A-63 und A-64). Anschließend konstruiert er ein requested_credentials-Objekt durch Mapping der fünf Attribute-Referents und des Predicate-Referents auf die Credential-ID, wobei Attribute mit „revealed: true“ gekennzeichnet werden. Der Versand der Presentation erzeugt einen ZKP, der kryptographisch beweist, dass der Holder ein Credential mit den geforderten Attributen besitzt, ohne die unrevealed Attribute offenzulegen (Listing A-65 und A-66).

Der Verifier empfängt die Presentation (State done, verified: true) und extrahiert die revealed Attributes durch dreistufiges Mapping, wodurch fünf offengelegte Attribute extrahiert werden (Listing A-68). Die drei Identitätsattribute bleiben geschützt und werden als „NICHT offengelegt (Zero-Knowledge-Proof)“ ausgewiesen, wodurch Privacy by Design gemäß DSGVO Art. 25 realisiert wird. Die Blockchain-basierte Revocation-Prüfung validiert den Credential-Status durch Vergleich des Indy-Proof-Timestamps mit dem Revocation-Registry-Delta auf dem Ledger. Die Zeitgültigkeitsprüfung vergleicht einen aktuellen Epoch-Timestamp mit den extrahierten Zeitgrenzen, wobei die Bedingung epoch_valid_from <= current_epoch <= epoch_valid_until die zeitliche Gültigkeit bestätigt. Die finale Zugriffsentscheidung kombiniert drei Validierungsergebnisse (not is_revoked AND is_time_valid AND has_required_clearance) und führt im KRITIS-Szenario zum Ergebnis „ZUGANG GEWÄHRT“.

6.1.5 Credential Revocation

Die funktionale Anforderung FR5 fordert, dass das System die Ungültigkeitserklärung ausgestellter Credentials mit kryptographischer Verifikation durch Verifier ermöglichen muss. Die Erfüllung wird anhand eines dreiphasigen Revocation-Workflows demonstriert, welcher Registry-Management, Two-Phase-Revocation sowie Non-Revocation-Proof Verification umfasst.

Phase 1 implementiert die Revocation-Registry-Verwaltung durch den Issuer. Der Abruf aktiver Registries (Listing A-70) zeigt zwei Registries mit 100 Credential-Kapazität. Jede Registry enthält die Metadaten `rev_reg_id` (eindeutiger Identifier), `tails_hash` (kryptographischer Hash für Non-Revocation-Proofs) und `tails_local_path` (Speicherort der Tails-File für Accumulator-basierte Revocation nach CL-Signature-Schema).

Phase 2 realisiert die Two-Phase-Revocation durch Staging und Ledger-Publishing. Die Staging-Phase (Listing A-72) referenziert das Credential via `rev_reg_id` und `cred_rev_id` und bestätigt mit dem Status „Pending“, dass die Revocation lokal gestaged ist. Die Publishing-Phase (Listing A-74) führt die Ledger-Transaktion aus (Typ 114 REVOC_REG_ENTRY) mit Accumulator-Updates und revokierten Credential-IDs.

Phase 3 validiert das Revocation-Enforcement. Das Holder-Wallet zeigt nach Revocation „Revoked Status: True“ (Listing A-75). Der Holder wählt das revokierte Credential aus, versucht einen Non-Revocation-Proof zu generieren, scheitert jedoch, da der Accumulator das Credential als revoked markiert (Listing A-76, Listing A-77, Listing A-78). Die finale Verifier-Entscheidung (Listing A-79) zeigt „Verified: false“ und „ZUGANG VERWEIGERT“, da trotz zeitlicher Gültigkeit und erfülltem ZKP-Predicate die Revocation-Prüfung dominiert.

6.1.6 Credential Deletion

Die funktionale Anforderung FR6 fordert, dass Holder Credentials lokal aus ihrem Wallet entfernen können, wobei diese Operation nur die lokale Datenhaltung betrifft und vom Ledger-basierten Revocation-Mechanismus zu unterscheiden ist.

Phase 1 implementiert das Pre-Deletion Inventory. Der Abruf aller im Holder-Wallet gespeicherten Credentials (Listing A-81) zeigt ein Credential mit „Revoked: True“. Dieses wurde zuvor via Ledger-Revocation ungültig erklärt, persistiert jedoch weiterhin im lokalen Wallet. Die referent-Identifier werden für die nachfolgende Deletion-Phase erfasst.

Phase 2 realisiert die Credential Deletion durch iterative Entfernung aller erfassten Credentials (Listing A-81). Für jeden credential_id wird via DELETE die lokale Wallet-Entfernung ausgeführt, mit erfolgreicher Deletion bestätigt: „Gelöscht: 1/1“. Diese Phase demonstriert die Holder-Autonomie über lokale Wallet-Daten ohne Issuer-Interaktion oder Ledger-Transaktion.

Phase 3 validiert die Deletion-Enforcement. Ein erneuter Abruf bestätigt das leere Wallet (Listing A-83). Lokale Deletion entfernt das Credential aus Holder-Verfügungsgewalt, jedoch bleibt die Ledger-basierte Revocation-Historie intakt, womit KRITIS-konforme Audit-Trails mit Privacy-wahren Holder-Rechten vereinbart werden. Diese Differenzierung ist kritisch für das SSI-Sicherheitsmodell.

6.2 Validierung der KRITIS-Compliance-Anforderungen

6.2.1 Einhaltung spezifischer Parameter-Sets für ML-DSA

Die Compliance-Anforderung zur Einhaltung BSI-konformer ML-DSA Parameter-Sets (NIST Security Strength Category 3 oder 5) wird durch strategische Verwendung der zwei Sicherheitsstufen ML-DSA-65 (Category 3) für operationale Signaturen sowie ML-DSA-87 (Category 5) für die Root CA erfüllt. Das finale Artefakt implementiert ML-DSA in drei Schichten. In der ersten Schicht werden TLS 1.3 Server-Zertifikate für alle fünf nginx Sidecar-Proxies (Issuer/Holder/Verifier, von-network Webserver, Tails Server) mittels ML-DSA-65 signiert (Listing A-5, Listing A-6, Listing A-8), während die Root CA mit ML-DSA-87 geschützt ist (Listing A-2). In der zweiten Schicht werden die did:peer:4 Signing Keys mit ML-DSA-65 generiert (Listing A-25). In der dritten Schicht nutzt DIDComm v1 Authcrypt Message-Signierung ML-DSA-65 via LibOQS (Listing A-13) für Sender-Authentifizierung.

Die operationale Integration wird durch Abbildung A-9 demonstriert, welche die TLS 1.3 Verbindung mit ML-DSA-65 signiertem Server-Zertifikat validiert, sowie durch die Wallet-Übersicht (Listing A-56), in welcher alle drei SSI-Agenten key_type ML-DSA-65 für did:peer:4 DID nutzen.

6.2.2 Einhaltung spezifischer Parameter-Sets für ML-KEM

Diese Anforderung wird durch systemweite Implementierung von ML-KEM-768 (Category 3) erfüllt. Das finale Artefakt implementiert ML-KEM-768 in zwei Schichten. In der ersten Schicht nutzt TLS 1.3 Key Exchange für alle fünf nginx-PQC-Sidecar-Proxies ML-KEM-768 konfiguriert via „DEFAULT_GROUPS=X25519MLKEM768mlkem768x25519“ in

den Docker-Infrastruktur-Definitionen (Listing A-5, Listing A-6, Listing A-8) sowie via `ssl_ecdh_curve X25519MLKEM768` in den nginx-Konfigurationen (Listing A-4). In der zweiten Schicht wird `did:peer:4` Key Agreement für Agent-to-Agent Verschlüsselung mit ML-KEM-768 realisiert (Listing A-25), womit DIDComm-Nachrichten Post-Quantum-resistant verschlüsselt werden.

Die operationale Integration wird durch Abbildung A-9 demonstriert, welche die TLS 1.3 Verbindung mittels der „Negotiated TLS 1.3 group: X25519MLKEM768“ validiert, sowie durch die Wallet-Übersicht (Listing A-56), in welcher alle drei SSI-Agenten `key_agreement_algorithm` ML-KEM-768 für `did:peer:4` DID nutzen.

6.2.3 Implementierung hybrider Schlüsseleinigung

Die BSI-Anforderung zur hybriden Schlüsseleinigung (Kombination klassisches Verfahren mit PQC-KEM) wird durch den X25519+ML-KEM-768-Hybrid-Modus erfüllt. Das finale Artefakt implementiert hybride Schlüsseleinigung systemweit via „`DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519`“ in allen Docker-Infrastruktur-Definitionen (Listing A-5, Listing A-6, Listing A-8), wobei die Priorisierung X25519MLKEM768 als primären Hybrid-Modus sicherstellt. Die TLS 1.3 Verbindungen aller fünf nginx-PQC-Sidecar-Proxies kombinieren X25519 mit ML-KEM-768 (Listing A-4). Zusätzlich implementiert `did:peer:4` hybride Key Agreement zwischen X25519- und ML-KEM-768-Keys aus DID Documents für DIDComm Message Encryption (Listing A-25). Die Fallback-Strategie `mlkem768:x25519` gewährleistet Interoperabilität mit Peers ohne Hybrid-Unterstützung, wobei reine PQC-Verschlüsselung via ML-KEM-768 Vorrang vor klassischem X25519 hat. Diese Architektur entspricht BSI (2025b, Kap. 2.2, 2.4) zur Absicherung gegen kryptanalytische Angriffe sowohl im klassischen als auch im Quantum-Computing-Bereich.

6.2.4 Bevorzugte Verwendung von TLS 1.3

Die BSI-Empfehlung zur vorrangigen Verwendung von TLS 1.3 wird durch systemweite TLS 1.3-Enforcement erfüllt. Das finale Artefakt erzwingt TLS 1.3 in allen fünf nginx-PQC-Sidecar-Proxies via „`ssl_protocols TLSv1.3;`“ in den Konfigurationsdateien (Listing A-4). Abbildung 23 demonstriert das Fehlschlagen des TLS 1.2 Verbindungsversuchs zum Issuer-Agenten.

Abbildung 23*TLS 1.3 Enforcement blockiert Legacy-Verbindungsversuch*

```
ferris@blockchain-ssi-pqc:~$ openssl s_client -connect host.docker.internal:8021 -tls1_2
Connecting to 172.17.0.1
CONNECTED(00000003)
4092659706720000:error:0A00042E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version:ssl/record/rec_layer_s3.c:916:SSL alert numbe
r 70
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 220 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Protocol: TLSv1.2
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
Protocol : TLSv1.2
Cipher   : 00000
Session-ID:
Session-ID-ctx:
Master-Key:
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1765391194
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
---
ferris@blockchain-ssi-pqc:~$
```

Anmerkung. Eigene Darstellung.

6.2.5 Protokollierung Sicherheitsrelevanter Ereignisse

Das finale Artefakt implementiert Protokollierung auf drei integrierten Ebenen zur Gewährleistung eines unveränderlichen Audit-Trails. ACA-Py Agent-Level Logging protokolliert Authentifizierungsversuche, Zustandsübergänge und Fehlerzustände via –log-level info (Listing A-8), wobei kryptographische Events wie ML-DSA-65 Signature Verification und DIDComm-Decryption dokumentiert werden (Listing A-84). nginx Access und Error Logs auf allen fünf Sidecar-Proxies erfassen TLS-Handshakes und HTTP-Requests (Listing A-5, Listing A-6), wobei Request-Body-Buffering für PQC-Schlüsselmaterialien protokolliert wird (Listing A-85). Die von-network Ledger Transaction Logs erfassen alle Blockchain-Operationen (NYM-, SCHEMA-, CRED_DEF-, REVOC_REG_ENTRY-Transaktionen) mit Zeitstempeln und Sequence Numbers, womit ein kryptographisch verifizierter, unveränderlicher Audit-Trail für KRITIS-Compliance-Anforderungen realisiert wird (Listing A-40, Listing A-74).

6.2.6 Logische Netzsegmentierung

Das finale Artefakt implementiert strikte Netzwerk-Segmentierung mittels dedizierter Docker Networks zur Isolation kritischer Infrastruktur-Komponenten. Die SSI-Agenten nutzen isolierte Networks (hope-issuer, hope-holder, hope-verifier) pro Agent, wobei nur zugehörige Sidecar-Proxies Zugriff haben (Listing A-8). Das Shared Network „von_sidecarproxy“ verbindet ausschließlich die nginx-PQC-Sidecar-Proxies untereinander für Agent-to-Agent Kommunikation, während interne Agent-Container isoliert bleiben. Die Blockchain-Nodes operieren im dedizierten „von-network“ mit separatem sidecarproxy Network für Webserver-Zugriff (Listing A-5), und der Tails Server nutzt ein isoliertes tails-server Network mit kontrolliertem Zugang via „von_sidecarproxy“ (Listing A-6). Diese mehrstufige Segmentierung, visualisiert in Abbildung 14 und Abbildung A-13, realisiert strenge Netzwerk-Isolation gemäß ISO/IEC (2022, Control A.8.22).

6.2.7 Datenschutz durch Technikgestaltung (Privacy by Design)

Die DSGVO-Anforderung zu Privacy by Design (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 25) wird durch architektonische Trennung von öffentlichen Identifikatoren und personenbezogenen Daten erfüllt. Das finale Artefakt realisiert eine konsequente Off-Chain-Architektur, in der personenbezogene Daten (Name, Organisation, Sicherheitsfreigabe) ausschließlich in verschlüsselten VC lokal im Holder-Wallet persistiert werden (Listing A-60), während Ledger-Transaktionen nur kryptographische

Identifikatoren (Schema-IDs, Credential-Definition-IDs, Revocation-Registry-IDs) und Public Keys enthalten (Listing A-40, Listing A-74). ZKPs ermöglichen selektive Offenlegung. Der Verifier erhält nur angeforderte Attribute, während sensible Daten geschützt bleiben, und Predicate-basierte Proofs ($\text{security_clearance_level} \geq 2$) erfolgen ohne Offenlegung exakter Werte (Listing A-79). Diese Architektur stellt Privacy by Default sicher, da personenbezogene Daten dezentral beim Holder verbleiben und nur kryptographisch verifizierbare Proofs ausgetauscht werden.

6.2.8 Grundsatz der Datenminimierung

Das finale Artefakt addressiert Datenminimierung durch drei komplementäre Mechanismen. Pairwise did:peer:4 DID eliminieren globale Identifikatoren. Für jede Agent-to-Agent Connection wird eine dedizierte DID generiert (Listing A-56), wodurch Transaktionskorrelation über verschiedene Verifier unterbunden wird und ein kompromittierter Verifier keine Aktivitäten des Holders bei anderen Verifiern nachverfolgen kann. Selective Disclosure in Proof Presentations offenbart ausschließlich angeforderte Attribute (`cert_type`, `facility_type`, `epoch_valid_from/until`, `role`), während Identitätsdaten (`first_name`, `name`, `organisation`) unrevealed bleiben (Listing A-79), wodurch nur zweckgebundene Minimaldaten übermittelt werden. Predicate-basierte ZKPs reduzieren Datenoffenlegung weiter. Der Holder beweist kryptographisch $\text{security_clearance_level} \geq 2$ ohne Preisgabe der exakten Stufe (Listing A-79). Diese mehrstufige Datenminimierung verhindert Profilbildung und Datensammlung, womit DSGVO-konforme Zweckbindung (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 5) technisch durchgesetzt wird.

6.2.9 Recht auf Löschung

Die DSGVO-Anforderung zum Recht auf Löschung (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 17) wird durch strikte Trennung von Ledger-Identifikatoren und Wallet-Daten erfüllt. Das finale Artefakt realisiert vollständige Löscharkeit personenbezogener Daten durch lokale Credential-Deletion via `DELETE /credential/<credential_id>`, wobei der sich im Holder-Wallet befindliche Credential (`<credential_id>`) gelöscht wird und damit alle personenbezogenen Attribute (Name, Organisation, Sicherheitsfreigabe) irreversibel verloren sind (Listing A-80). Crypto-Shredding durch Wallet-Destruction gewährleistet kryptographische Unlesbarkeit aller Credential-Daten. Die Vernichtung des Wallet-Keys führt zur Unmöglichkeit der Entschlüsselung, selbst wenn Backups existieren (Listing A-81). Während Blockchain-Transaktionen (Schema-IDs, Cred-Def-IDs, Revocation-Entries) unveränderlich auf dem Ledger verbleiben, enthalten diese

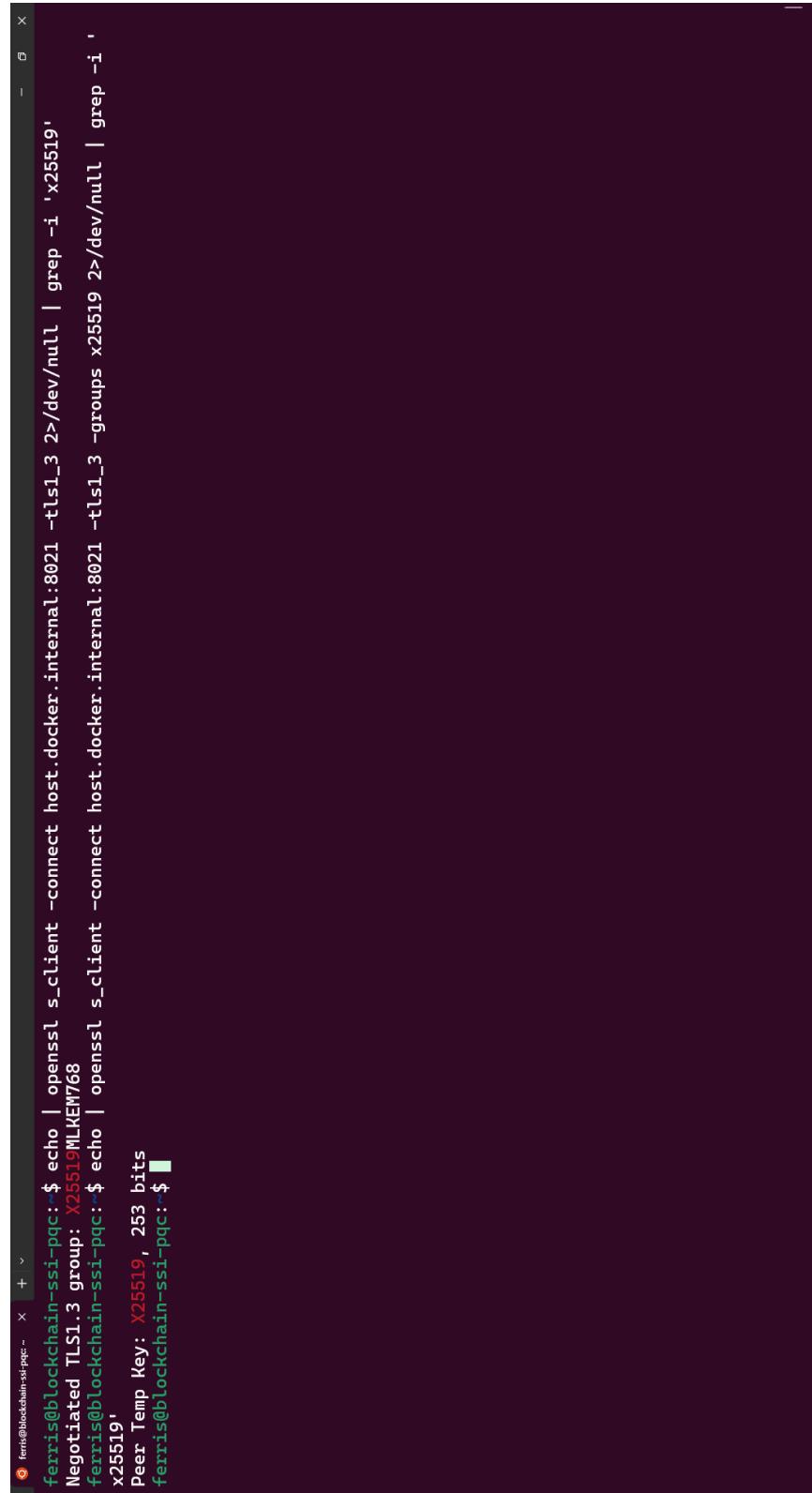
per Design keine personenbezogenen Daten, sondern nur kryptographische Identifier (Listing A-81). Diese Architektur gewährleistet vollständige Löschung personenbezogener Daten bei gleichzeitiger Audit-Trail-Integrität.

6.3 Validierung der Kryptoagilität

6.3.1 Transportlayer

TLS 1.3 realisiert Kryptoagilität durch orthogonale Aushandlung von Cipher Suite, Schlüsselaustauschverfahren und Signaturalgorithmen (Rescorla, 2018, S. 26), wodurch jeder Mechanismus unabhängig modifiziert werden kann. Die `supported_groups`-Erweiterung ermöglicht es Endpunkten, Schlüsselaustauschverfahren unabhängig vom symmetrischen Verschlüsselungsverfahren auszuhandeln (Rescorla, 2018, S. 47). Diese Architektur adressiert die Kryptoagilitätseigenschaften Extensibility und Removability nach Mehrez und El Omri (2018, S. 102).

Die Implementierung nutzt hierfür eine konfigurationsbasierte Algorithm-Fallback-Chain via „`DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024`“ (Listing A-3), die primär hybride Verfahren nutzt und automatisch auf rein klassische Verfahren (`x25519`) bei Inkompatibilität zurückfällt. Dies erfüllt die Fungibility-Anforderung nach Mehrez und El Omri (2018, S. 102) und ermöglicht Algorithmenauswahl in Echtzeit basierend auf Sicherheitsfunktion (Cyber Resilience Workshop Series Committee et al., 2017, S. 19). Abbildung 24 demonstriert den kryptoagilen Fallback-Prozess.

Abbildung 24*TLS 1.3 Kryptoagiler Fallback von X25519ML-KEM-768 zu X25519*

A screenshot of a terminal window titled "ferris@blockchain-ssi-pqc: ~". The window displays a command-line session for testing TLS 1.3 negotiation. The user runs "openssl s_client -connect host.docker.internal:8021 -tls1_3 2>/dev/null | grep -i 'x25519'". The output shows the negotiation of a group named "X25519MLKEM768", which is identified as a negotiated TLS 1.3 group. The user then runs "openssl s_client -connect host.docker.internal:8021 -tls1_3 -groups x25519 2>/dev/null | grep -i 'x25519'". The output shows the negotiation of a peer temporary key named "X25519", which is described as having 253 bits.

```
ferris@blockchain-ssi-pqc: ~
ferris@blockchain-ssi-pqc:~$ echo | openssl s_client -connect host.docker.internal:8021 -tls1_3 2>/dev/null | grep -i 'x25519'
Negotiated TLS 1.3 group: X25519MLKEM768
ferris@blockchain-ssi-pqc:~$ echo | openssl s_client -connect host.docker.internal:8021 -tls1_3 -groups x25519 2>/dev/null | grep -i 'x25519'
Peer Temp Key: X25519, 253 bits
ferris@blockchain-ssi-pqc:~$
```

Anmerkung. Eigene Darstellung.

6.3.2 Applikationslayer

Die Applikationsebene nutzt die Plugin-Architektur von ACA-Py zur Algorithmenauswahl ohne Modifikation des Kerncode, entsprechend dem Open-Closed-Prinzip nach Martin (2003, S. 99). Das Plugin implementiert einen metadatengesteuerten Ansatz. Via Parameter „metadata: key_type: ed25519“ kann der Schlüsseltyp explizit spezifiziert werden (Listing A-86). Der OutOfBandManager evaluiert in `create_did_peer_4_conditional_pqc` (Listing A-20) den key_type-Parameter. Wird „ed25519“ erkannt, delegiert das Plugin zur ursprünglichen ACA-Py-Implementierung für vollständigen Fallback auf klassische Kryptografie, fehlt die Spezifikation, aktiviert das System standardmäßig ML-DSA-65 und ML-KEM-768.

Die erfolgreiche Validierung zeigt Listing A-87. Eine ED25519-basierte Verbindung etabliert sich mit Status „active“, was korrekte DID Exchange-Durchführung mit klassischen Algorithmen bestätigt. Listing A-89 verifiziert persistierte Kryptografie. Wallet-DID zeigen konsistent „key_type: ed25519“, wobei Abwesenheit von PQC-Metadaten (`pqc_enabled`, `signature_algorithm`, `kem_verkey`) bestätigt, dass das Plugin keine Eingriffe im ED25519-Workflow vornahm.

7 Ergebnisse und Diskussion

7.1 Beantwortung der Forschungsfragen

Die vorliegende Arbeit adressiert durch DSR Methodologie drei zentrale Forschungsfragen an der Schnittstelle von SSI, PQC und KRITIS. Die in Kapitel 6 durchgeführte summative Evaluation unter Anwendung des FEDS-Frameworks demonstriert, dass das entwickelte Artefakt diese Fragen nicht nur theoretisch sondern empirisch durch kontrollierte Evaluation beantwortet.

FF1 – Systemarchitektur & Compliance: Wie kann ein blockchain-basiertes SSI-System unter Einsatz von PQC gestaltet werden, um die regulatorischen und technischen Anforderungen von KRITIS nachhaltig zu erfüllen?

Diese Forschungsfrage wird durch die Konzeption und Implementierung einer zweischichtigen Architektur beantwortet, die quantenresistente Kryptografie orthogonal und nicht-invasiv in bestehende SSI-Infrastrukturen dem mehrschichtigen Sicherheitsansatz „Defense in Depth“ nach Alsaqour et al. (2021, S. 242–243) folgend integriert.

Die erste Schicht realisiert Quantensicherheit auf der Transportebene (Data-In-Motion) mittels eines Sidecar-Proxy Patterns mit nginx, das TLS 1.3 als Protokoll-Standard durchsetzt und ein hybrides Schlüsseleinigungsverfahren (X25519 + ML-KEM-768) implementiert. Parallel dazu werden digitale Zertifikate mit ML-DSA-65-Signaturen ausgestellt und validiert, was die Post-Quantum-Authentifizierung von Netzwerk-Endpunkten sicherstellt. Diese nicht-invasive Architektur erfüllt die Transition-Anforderung nach Backward-Kompatibilität (Mamatha, Dimri & Sinha, 2024, S. 3). SSI-Agenten (ACA-Py), Blockchain-Knoten (Hyperledger Indy) und Wallet-Applikationen benötigen keine Modifikation ihrer Anwendungslogik, da die Quantensicherung transparent auf der Netzwerk-Infrastruktur-Ebene erfolgt.

Die zweite Schicht erweitert die Quantensicherheit auf die Applikationsebene (Data-At-Rest und Credential-Verarbeitung) durch native PQC-Integration in Hyperledger ACA-Py. Mittels einer Monkey-Patching-Strategie werden zur Laufzeit kritische Komponenten überschrieben. Die DID-Verarbeitung ermöglicht es dezentrale Identifikatoren mit ML-DSA-65-Signaturen zu erzeugen und zu validieren. Gleichzeitig unterstützt die DIDComm-Envelope-Verarbeitung ML-KEM-768-basierte Schlüsselkapseln für die Nachrichtenverschlüsselung. Diese duale Implementierung folgt dem Sicherheitsprinzip der Defense in Depth, bei welcher gemäß Alsaqour et al. (2021, S. 242–243) durch mehrschichtige Sicherheitsmaßnahmen ein Ausfallschutz garantiert wird, sodass das Versagen einer einzelnen Schicht nicht zum Systemversagen führt.

Die Validierung aller neun Compliance-Anforderungen (CR1–CR9) in Kapitel 6.2 demonstriert empirisch, dass die Architektur sowohl die kryptografischen Parameter für ML-DSA-65 und ML-KEM-768 (BSI, 2025b, Kap. 2.4.3, 5.3.4.2), als auch die organisatorischen Anforderungen der DSGVO an Privacy by Design (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 25), den Grundsatz der Datenminimierung (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 5) und das Recht auf Löschung (Das europäische Parlament und der Rat der europäischen Union, 2016, Art. 17) erfüllt (CR7–CR9). Insbesondere werden die Anforderungen zur strikten Netzsegmentierung (ISO/IEC, 2022, Control A.8.22) (CR6) und die Protokollierung sicherheitsrelevanter Ereignisse (BSI, 2024, Nr. 101, 103) (CR5) vollständig operationalisiert. Das Artefakt demonstriert somit, dass quantenresistente Kryptografie für die Anwendungsdomäne KRITIS nicht nur ein technisches, sondern ein systemisches Problem ist, das Compliance-, Architektur- und Governance-Aspekte simultan adressieren muss.

FF2 – Algorithmenauswahl & Sicherheitsbewertung: Welche PQC-Algorithmen eignen sich für die Integration in SSL-Systeme hinsichtlich Sicherheit und Interoperabilität, insbesondere im Kontext von KRITIS?

Diese Forschungsfrage wird durch die empirische Validierung der NIST-standardisierten „Security-Kategorie 3“ Algorithmen ML-DSA-65 (National Institute of Standards and Technology, 2024a, Tab. 1) und ML-KEM-768 (National Institute of Standards and Technology, 2024b, S. 14) beantwortet. Die bewusste Auswahl dieser Parameter-Sets entspricht der direkten Empfehlung des BSI (BSI, 2025b, S. 41, 58).

Die empirische Evaluation in Kapitel 6.1 und Kapitel 6.2 zeigt, dass diese Algorithmen nicht primär durch intrinsische Sicherheitsdefizite limitiert sind, sondern durch technische Integrationschallenges in bestehende Hyperledger-Aries-Infrastrukturen, die originär auf elliptischen Kurven (X25519, Ed25519) basieren und keine native PQC-Unterstützung bieten (Badertscher, Banfi & Diaz, 2024, S. 4733, 4735). Beide Verfahren basieren auf der rechnerischen Schwierigkeit gitterbasierter Probleme, insbesondere dem Shortest Vector Problem und dem Module Learning With Errors Problem, deren mathematische Härte rigoros analysiert ist (Zong, 2025, S. 1). Die Infrastruktur-Kompatibilität adressiert die Arbeit durch drei komplementäre Mechanismen.

Erstens ermöglicht TLS 1.3-Algorithmen-Aushandlung es Endpunkten, zwischen hybriden Cipher-Suites und rein klassischen Verfahren einen Fallback zu nutzen (Rescorla, 2018, S. 26), was die schrittweise Migration für heterogene Deployments vereinfacht. Dies wird durch eine konfigurationsbasierte Fallback-Chain in Listing A-4 realisiert, die automatisch auf rein X25519 zurückfällt.

Zweitens detektiert die erweiterte Envelope-Verarbeitung anhand von Multicodec-Präfixen automatisch, ob PQC oder klassische Algorithmen erforderlich sind. Der OutOfBandManager evaluiert hierfür den keytype-Parameter via `create_did_peer_4_conditional_pqc` und delegiert bei `ed25519` zur ursprünglichen ACA-Py-Implementierung, während das System standardmäßig ML-DSA-65 und ML-KEM-768 aktiviert (Listing A-20, Listing A-86 und Listing A-87). Dies ist eine notwendige Erweiterung, da DIDComm-v1 keine native Kryptografiebilität bietet und ausschließlich auf X25519 für Key Exchange sowie Ed25519 für Signing festgelegt ist (Badertscher, Banfi & Diaz, 2024, S. 4735). Bedingt dadurch, dass DIDComm keine Algorithmen-Aushandlung vorsieht (Badertscher, Banfi & Diaz, 2024, S. 4733) stellt die PQC-Integration eine nicht-standardkonforme Erweiterung dar, die durch Monkey-Patching der Verarbeitungslogik realisiert werden muss (Listing A-20 und Listing A-86).

Drittens schafft die Multikey-Kodierung mittels Multicodec-Registry die kryptographische Identifikation neuer Algorithmen ohne Schema-Änderungen im DID-Dokument. Das Aries-Askar-Wallet kodiert ML-DSA-65- und ML-KEM-768-Schlüsselpaare mit den definierten Multicodec-Präfixen Listing A-16 und konvertiert diese zu Multikey-Strings Listing A-17, die in der SQLite-Datenbank persistiert werden, ohne dass die DID-Dokumentstruktur angepasst werden muss.

Die Hybrid-Strategie, die klassische und quantenresistente Algorithmen kombiniert, ist ein Best-Practice für die aktuelle Übergangswelt (BSI, 2025b, S. 22). Die Kombination von X25519 (ECC, etabliert sicher) mit ML-KEM-768 (PQC, mathematisch neu) garantiert, dass die Gesamtsicherheit mindestens so stark wie der stärkere der beiden Algorithmen ist.

FF3 – Kryptografische Agilität: Welche kryptografischen Agilitätsmechanismen sind erforderlich, um zukünftige PQC-Algorithmenupdates ohne Systemunterbrechung zu ermöglichen?

Diese Forschungsfrage wird durch die Implementierung zweier orthogonaler, schichten-spezifischer Agilitätsmechanismen beantwortet, die das Open-Closed-Prinzip nach Martin (2003, S. 99) realisieren und damit die Eigenschaften Extensibility, Removability und Functionality nach Mehrez und El Omri (2018, S. 102–103) operationalisieren. Die Architektur adressiert kryptografische Agilität nicht als nachträgliche Erweiterung, sondern als fundamentales Designprinzip, das durch systematische Separation of Concerns auf beiden Schichten verankert wird.

Auf der Transportebene wird Agilität durch die Protokoll-inhärente Aushandlungslogik von TLS 1.3 realisiert. Durch die Konfiguration einer präferierten Cipher-Suite-Reihenfolge im

Sidecar-Proxy wird ein automatischer Fallback-Mechanismus etabliert, der hybride Verfahren (X25519+ML-KEM-768) priorisiert, jedoch transparent auf klassische Verfahren (X25519) zurückfällt, sofern die Gegenseite keine PQC-Unterstützung signalisiert. Diese Entkopplung ermöglicht den Austausch kryptografischer Primitive durch reine Konfigurationsanpassungen ohne Rekomplikierung oder Systemunterbrechung, was die Anforderungen an Extensibility und Removability nach Mehrez und El Omri (2018, S. 102) erfüllt.

Auf der Applikationsebene adressiert das entwickelte Plugin „pqc_didpeer4_fm“ die fehlende native Agilität von DIDComm-v1 (Badertscher, Banfi & Diaz, 2024, S. 4733) durch eine modulare Architektur nach dem Software Stability Model von Hamza (2005, S. 1–2). Die Stratifizierung in stabile abstrakte Operationen (EBT), adaptierbare Geschäftsobjekte (BO) und volatile Implementierungen (IO) erlaubt es, Algorithmen austauschbar zu gestalten. Mittels Monkey-Patching (IO) werden zur Laufzeit kritische ACA-Py-Funktionen überschrieben, während die BO-Schicht die Koexistenz verschiedener Schlüsseltypen (z. B. Ed25519 neben ML-DSA-65) ohne Schema-Änderungen an DID-Dokumenten ermöglicht.

Die empirische Evaluation in Kapitel 6.3 bestätigt, dass durch diese Mechanismen ein heterogener Mischbetrieb möglich ist. Das System etabliert erfolgreich PQC-gesicherte Verbindungen, während es für Legacy-Agenten automatisch und unterbrechungsfrei auf klassische Algorithmen (Ed25519) zurückfällt (Listing A-87).

7.2 Kritische Reflexion

Die vorliegende Arbeit demonstriert erfolgreich, dass die Entwicklung eines blockchain-basierten SSI-Prototypen unter Integration von PQC technisch machbar und unter Laborbedingungen validierbar ist. Die iterative Anwendung der DSR-Methodik sowie die umfassende Evaluation gegen funktionale und Compliance-Anforderungen stellen einen substantiellen Beitrag dar, der die Forschungslücke zwischen abstrakten PQC-Standards und ihrer praktischen Umsetzung in dezentralen Identitätssystemen verringert.

Gleichwohl verdient eine kritische Reflexion bestimmte Designentscheidungen zu beleuchten, um sowohl die Stärken als auch die bewussten Limitation dieser Arbeit transparent darzulegen. Die Nutzung von Monkey-Patching zur Laufzeitmodifikation von ACA-Py-Klassen ist pragmatisch und methodisch im Kontext eines akademischen Prototypen gerechtfertigt, da das Framework zum Erhebungszeitpunkt keine nativen Mechanismen für flexible Kryptografieauswahl auf der Applikationsebene bietet (Badertscher, Banfi & Diaz, 2024, S. 4733, 4735). Diese Implementierungstechnik erlaubt es, PQC-Algorithmen zu

injizieren, ohne den ursprünglichen Quellcode zu verändern. Für eine potenzielle Produktionisierung wäre jedoch eine saubere Architekturrefaktorisierung oder die Integration nativer Krypto-Plugin-Schnittstellen erforderlich, damit Betreiber die im KRITIS-Kontext empfohlene Berücksichtigung von IT-Sicherheit bereits im Entwicklungsprozess (Security by Design), die Transparenz über Sicherheits- und Verfügbarkeitseigenschaften sowie die Dokumentation von Designprozessen und implementierten Maßnahmen adäquat abbilden können (BSI, 2022, S. 4, 8–12). Dies stellt weniger eine Schwäche des Prototypen, welcher im begrenzten Rahmen dieser Masterarbeit entstanden ist, dar, sondern verdeutlicht vielmehr die Notwendigkeit, dass die Hyperledger-Community zukünftig Kryptoagilität als Design-Prinzip in ihre Kernarchitktur integriert.

Ein zweiter Aspekt betrifft die Protokollebene und die Wahl von DIDComm v1. Diese Entscheidung beruht auf der zum Zeitpunkt der Entwicklung verfügbaren Stabilität und Reife der Bibliotheken. Da DIDComm v1 jedoch kryptografisch strikt auf elliptische Kurven normiert ist, musste die Kryptoagilität durch eine hybride Kapselung auf der Transport-Ebene (TLS 1.3 mit Sidecar-Proxies) realisiert werden. Das Resultat ist architektonisch konsistent und funktioniert zuverlässig in der isolierten Laborumgebung. Die Interoperabilität mit Standard-SSI-Wallets, die diese Erweiterungen nicht verstehen, ist jedoch begrenzt. Die Arbeit trägt damit nicht nur zu einem funktionierenden Artefakt bei, sondern identifiziert auch, wo Standards weiterentwickelt werden müssen, um zukünftige Anforderungen zu erfüllen.

Bezüglich der Evaluationsstrategie wurde bewusst der Fokus auf den Nachweis der „Technical Feasibility“ durch rigorose Demonstration der „Efficacy“ gelegt, während Efficiency-Metriken (Performance, Latenz, Speicheroverhead) nicht erhoben wurden (Kapitel 3.3). Diese Abgrenzung ist zum einen durch den begrenzten Rahmen der Masterarbeit, zum anderen methodisch begründet, da nach Rushby (2004, S. 110–111) bei kritischen Systemen die Integrität und die Vermeidung schwerwiegender Fehlerzustände eine qualitative Sonderstellung einnehmen und somit Vorrang vor der Optimierung der regulären Betriebsleistung haben. Nichtsdestotrotz ist anzumerken, dass die größeren Schlüssel und Signaturen der PQC-Algorithmen (insbesondere ML-DSA und ML-KEM) in skalierten Blockchain-Szenarien zu Speicher- und Netzwerk-Engpässen führen könnten. Eine zukünftige Erweiterung dieser Arbeit sollte systematisch untersuchen, wie sich diese PQC-Overhead-Faktoren auf die Throughput und Latenz verteilter Ledger auswirken und welche Optimierungsstrategien möglich sind.

Schließlich ist eine Limitation in der Evaluationsdimension zu reflektieren. Die Validierung der Compliance-Anforderungen erfolgte primär auf technischer Ebene, etwa durch den Nachweis, dass ML-DSA-Schlüssellängen den NIST-Vorgaben entsprechen oder dass

TLS 1.3 erzwungen wird. Echte KRITIS-Compliance ist mehr als nur technische Validierung. Sie erfordert organisatorische Security Controls, die Rollen, Incident-Management-Prozesse und Audit-Verfahren systematisch integrieren (Alcaraz & Zeadally, 2015). Der entwickelte Prototyp demonstriert technische „Compliance Readiness“ durch die Validierung gegen die definierten KRITIS-Compliance-Anforderungen aus Tabelle 5, nicht aber die volle organisatorische Compliance. Dies ist jedoch eine erwartete Limitation einer akademischen Arbeit und keine Schwäche des Artefakts.

7.3 Wissenschaftliche und praktische Beiträge

Diese Arbeit leistet einen ersten umfassenden Beitrag zur Integration der drei bislang isoliert betrachteten Forschungsdomänen SSI, PQC und KRITIS in einem kohärenten und empirisch validierten Systemdesign. Die Forschungslandschaft adressiert die drei Domänen isoliert (Kapitel 1.2), während die Formulierung einer ganzheitlichen, quantensicheren SSI-Architektur speziell für KRITIS-Kontexte bislang nicht systematisch entwickelt wurde. Ein zentrales Ergebnis ist die Erkenntnis, dass quantenresistente Kryptografie nicht als bloße Kryptografie-Substitution verstanden werden darf, sondern als ein Architektur-Problem, das fundamentale Anforderungen an die Systemgestaltung stellt. Die Integration neuer Kryptographischer Primitive, die Gewährleistung von Interoperabilität zwischen PQC-fähigen und Legacy-Systemen, die Realisierung von Kryptoagilität sowie die Wahrung von Privacy-by-Design bei veränderten Signatur-Algorithmen erfordern spezifische technische Lösungsmuster wie Sidecar-Proxies für transparente TLS, Monkey-Patching für Applikations-Layer-Agilität, Multicodec-Extensibilität und sorgfältige Governance-Modelle.

Ein weiterer wissenschaftlicher Beitrag ist die formale Charakterisierung von schichtenspezifischer Kryptoagilität gemäß dem Software Stability Model nach Hamza (2005). Während Transport-Layer-Agilität durch TLS 1.3 mittels Mechanismen wie Cipher Suite Negotiation und Post-Handshake Key Updates (Rescorla, 2018, S. 26, 31, 76, 133) adressiert wird, erfordert Applikations-Layer-Agilität die Implementierung von Rückwärtskompatibilität auf mehreren Ebenen. Insbesondere müssen DIDComm-Envelope-Verarbeitung, Credential-Verifikation und Revocation-Mechanismen so gestaltet werden, dass Systeme mit unterschiedlichen Algorithmus-Generationen interoperieren können, ohne bestehende Verifikationsketten zu invalidieren. Das Plugin-Modell demonstriert, dass durch Separation von stabilen Abstraktionsschichten (EBT), adaptiven Konkretisierungsschichten (BO) und volatilen Implementierungsdetails (IO) nach Hamza (2005, S. 2) Algorithmen-Updates nicht als Breaking Changes kodifiziert werden müssen. Neue Algorithmen werden stattdessen als BO-Erweiterungen, unter Einhaltung der stabilen EBT-Interfaces, behandelt,

wodurch Kryptoagilität nicht länger als architektonische Belastung, sondern als natürliche Konsequenz strukturierter Modularität entsteht. Diese Erkenntnisse zur schichtenweisen Trennung von Stabilität und Volatilität sind generalisierbarerweise auf alle Systeme mit Kryptografie-Migrationserfordernissen übertragbar. Organisationen können diese architektonischen Muster als PQC-Adoptionspfade nutzen, da die Prinzipien der schichtenweisen Trennung und pluggingestützten Extensibilität domänenneutral sind. Das iterative DSR-Vorgehen demonstriert, dass formative Evaluation als ganzheitliche Methodik für technologische Migrationen in regulativ kontrollierten Kontexten einsetzbar ist.

Zu den praktischen Beiträgen zählt das entwickelte Artefakt, bestehend aus Sidecar-Proxy-Templates, ACA-Py-Plugin-Code, vollständiger Docker-Compose-Orchestrierung und ausführlichen Jupyter-Notebooks. Die Dokumentation basierend auf realistischen KRITIS-Szenarios ermöglicht es, die Technologie-Anwendbarkeit unabhängig zu bewerten. Über die bloße Bereitstellung des Artefakts hinaus leistet die modulare Architektur einen praktischen Beitrag durch die Ermöglichung inkrementeller Adoptionsstrategien, die der organisatorischen Realität entsprechen. Eine Organisation könnte zunächst nur die Sidecar-Proxy-Layer deployieren, um ihre bestehende SSI-Infrastruktur unmittelbar quantensicher zu gestalten, und später zur Applikations-Layer-Integration übergehen. Diese Strategie vermeidet Big-Bang-Deployments und ermöglicht gradueller, reversibel-testbare Übergänge, die dem organisatorischen Änderungsmanagement entsprechen. Das identifizierte Sidecar-Proxy-Muster ist nicht auf SSI-Szenarien begrenzt, sondern auf beliebige Microservice-Architekturen mit nicht-invasivem Sicherheitsanforderungsmanagement anwendbar. Die Plugin-basierte Applikations-Layer-Architektur zeigt, dass Kryptografie-Abstraktionen in beliebigen Frameworks mit Plugin-Mechanismen operationalisierbar sind, womit die Erkenntnisse auf andere Identitätsmanagementsysteme oder dezentrale Anwendungen transferierbar werden.

Das Prinzip „Defense in Depth“ von Alsaqour et al. (2021, S. 242–243) motivierte ein iteratives Design-Refinement mit wissenschaftlicher und praktischer Relevanz. Dieses Vorgehen spiegelt authentische Entwicklungsprozesse wider und fungiert als Blaupause für andere Projekte im Bereich der Kryptografie-Migration. Auf technologischer Ebene wird durch die Konstruktion des liboqs-Python-Bindings mit Multicodec-Integration gezeigt, dass moderne Kryptografiebibliotheken in bestehende Identitäts-Ökosysteme integrierbar sind. Die Implementierung der did:peer:4-DID-Methode mit PQC-Unterstützung erweitert die W3C-Spezifikationen um eine quantenresistente Variante, ohne die bestehenden ed25519-Verfahren zu beeinträchtigen. Diese Strategie folgt dem Prinzip adaptiver Modularität, bei dem neue Algorithmen als Erweiterungen bestehender Schnittstellen implementiert werden, anstatt fundamentale Architekturänderungen zu erzwingen.

8 Fazit und Ausblick

8.1 Zusammenfassung der Ergebnisse

Die vorliegende Masterarbeit entwickelt und evaluiert im Rahmen des DSR einen funktionsfähigen Prototypen für eine quantensichere dezentrale Identitätsverwaltung in KRITIS. Ausgangspunkt ist das Spannungsfeld zwischen den architektonischen Sicherheitsvorteilen von SSI und der Verwundbarkeit aktueller Implementierungen durch zukünftige Quantencomputer. Die systematische Literaturrecherche nach PRISMA-Standards zeigt auf, dass die praktische Integration NIST-standardisierter PQC unter gleichzeitiger Wahrung von Dezentralisierungsprinzipien und KRITIS-Compliance bislang eine Forschungslücke darstellt.

Die iterative Artefaktentwicklung liefert als zentrales architektonisches Ergebnis die Erkenntnis, dass zur Schließung dieser Lücke zwei komplementäre Integrationsmuster erforderlich sind. Auf der Transportebene ermöglicht ein Sidecar-Proxy-Ansatz mit PQC-zertifizierten TLS-1.3-Verbindungen eine transparente Quantenverschlüsselung ohne Änderungen an der SSI-Agent-Logik. Auf der Applikationsebene erlaubt ein Plugin-basiertes System mit Monkey-Patching die Integration von PQC-Operationen in den Credential-Lebenszyklus ohne invasive Code-Modifikationen.

Ein zweites wesentliches Ergebnis betrifft die Kryptoagilität. Die Masterarbeit verdeutlicht, dass langfristige Quantensicherheit weniger einen statischen Zustand als vielmehr die architektonische Fähigkeit erfordert, auf die dynamische Entwicklung kryptografischer Standards reagieren zu können. Dies wird technisch durch Abstraktionsschichten gelöst, welche den Austausch kryptografischer Subroutinen zur Laufzeit ermöglichen. Neue Algorithmen lassen sich so mittels Dockerfile-Modifikationen nahtlos integrieren, ohne die Verfügbarkeit der Systeme zu beeinträchtigen.

Ein drittes Kernresultat betrifft die KRITIS-Compliance. Die Arbeit operationalisiert regulatorische Vorgaben aus nationaler Cybersicherheit, Datenschutzrecht und internationalen Sicherheitsstandards in ein konsistentes technisches Anforderungsprofil. Die summative Evaluation bestätigt dessen vollständige Erfüllung, insbesondere durch hybride Kryptoverfahren zur Risikominimierung und granulare Audit-Mechanismen als Basis für SzA. Damit wird der Nachweis erbracht, dass Quantensicherheit und regulatorische Compliance auch innerhalb dezentraler Architekturen gemeinsam operationalisierbar sind.

In holistischer Betrachtung bestätigt die Masterarbeit die technische Realisierbarkeit einer quantenresistenten Transformation dezentraler Identitätssysteme für KRITIS. Die erarbeiteten Architekturmuster und Agilitätsmechanismen fundieren ein übertragbares Referenzmodell für die Praxis. Dieses befähigt Organisationen zur proaktiven Härtung ihrer Infrastrukturen, indem es einen validierten Pfad zur Quantensicherheit aufzeigt, der Dezentralisierungsprinzipien und regulatorische Compliance in Einklang bringt.

8.2 Zukünftige Forschungsrichtungen

Anknüpfend an den hier erbrachten Nachweis der technischen Machbarkeit fokussiert ein erster und dringlicher Forschungsstrang auf die quantitative Performance-Evaluation (Efficiency). Da Post-Quantum-Algorithmen wie ML-DSA und ML-KEM signifikant größere Schlüssel- und Signaturlängen aufweisen als klassische Verfahren, ist eine systematische Messung von Latenzzeiten, Durchsatzraten und Ressourcenverbrauch unter Last erforderlich. Zukünftige Studien sollten untersuchen, wie sich diese kryptografischen Overheads in ressourcenbeschränkten IoT-Umgebungen, die für kritische Infrastrukturen im OT-Bereich typisch sind, auf die Echtzeitfähigkeit der Kommunikation auswirken und welche Optimierungspotenziale durch Hardware-Beschleunigung realisierbar sind.

Ein zweites Forschungsfeld betrifft den Transfer der Evaluation von der kontrollierten Laborumgebung (Artificial Evaluation) in reale Anwendungsszenarien (Naturalistic Evaluation). Während dieser Prototyp in einer isolierten Entwicklungsumgebung mittels Docker validiert wurde, erfordert der produktive Einsatz in KRITIS die Untersuchung von Interdependenzen mit heterogenen Legacy-Systemen und komplexen Netzwerktopologien. Folgeforschung sollte sich daher auf Pilotierungen in realen KRITIS-Sektoren konzentrieren, um die Robustheit der Sidecar-Proxy-Architektur gegenüber realen Angriffsvektoren, Netzwerklatenzen und Ausfallszenarien empirisch zu validieren.

Schließlich erfordert die ganzheitliche Betrachtung eine Erweiterung des Forschungsfokus von der technischen Compliance Readiness hin zur organisatorischen Governance. Während die vorliegende Masterarbeit die technischen Voraussetzungen für Datenschutz und Angriffserkennung validiert hat, bedarf die prozessuale Verankerung dieser Artefakte in etablierte Sicherheitsmanagementsysteme weiterführender Untersuchung. Zukünftige Arbeiten sollten daher evaluieren, welche adaptierten Prozessmodelle und Audit-Verfahren notwendig sind, um dezentrale Identitätsarchitekturen und dynamische Kryptoagilität nahtlos und auditsicher in die betrieblichen Governance-Strukturen von KRITIS-Betreibern zu integrieren.

Anhang

Inhaltsverzeichnis des Anhangs

Anhang 1 Self-Sovereign Identity	104
Anhang 2 Systematische Literaturrecherche	105
Anhang 2.1 Erste Iteration vom 30. Mai 2025 (Exposé)	106
Anhang 2.2 Zweite Iteration vom 02. November 2025	168
Anhang 3 Erste Iteration der Artefaktentwicklung	203
Anhang 3.1 Framework- und Technologie-Auswahl	203
Anhang 3.1.1 DLT-Plattform	203
Anhang 3.1.2 SSI-Framework	203
Anhang 3.1.3 Kryptografiebibliothek	204
Anhang 3.1.4 Revocation-Infrastruktur	205
Anhang 3.1.5 Sidecar-Proxy	205
Anhang 3.2 Implementierung	206
Anhang 3.2.1 Setup der Entwicklungsumgebung	206
Anhang 3.2.2 Zertifikatserstellungsworkflow	209
Anhang 3.2.3 Dockerfile: Sidecar-Proxy nginx	211
Anhang 3.2.4 nginx_holder.conf	215
Anhang 3.2.5 DLT-Infrastruktur	218
Anhang 3.2.6 Revocation Registry	223
Anhang 3.2.7 Dockerfile: acapy-base	225
Anhang 3.2.8 SSI-Agenten	227
Anhang 3.2.9 Docker Orchestrierung der Gesamtarchitektur	234
Anhang 3.3 Formative Evaluation	257

Anhang 3.3.1	Cipher Mismatch der TLS-1.3-Verbindung des Webservers.....	257
Anhang 3.3.2	Eigenkomplilation eines PQC-Chromium-Browsers	259
Anhang 3.3.3	Validierung der Zertifikatskette und ML-DSA-Signaturen	260
Anhang 3.3.4	Validierung der TLS 1.3 Algorithmen-Aushandlung	262
Anhang 3.3.5	Validierung der Ledger-Initialisierung.....	264
Anhang 3.3.6	Validierung der ACA-Py API-Verfügbarkeit	267
Anhang 3.3.7	Validierung der Netzwerkisolation	270
Anhang 4	Zweite Iteration der Artefaktentwicklung.....	273
Anhang 4.1	ACA-Py-Plugin-Quellcode (pqc_didpeer4_fm)	273
Anhang 4.1.1	liboqs_wrapper.py	273
Anhang 4.1.2	pqc_peer4_creator.py.....	277
Anhang 4.1.3	pqc_peer4_resolver.py.....	279
Anhang 4.1.4	pqc_multicodec.py.....	281
Anhang 4.1.5	pqc_multikey.py.....	282
Anhang 4.1.6	pqc_didcomm_v1.py.....	284
Anhang 4.1.7	monkey_patches.py	293
Anhang 4.1.8	base_manager_patch.py	296
Anhang 4.1.9	askar_pqc_patch.py	310
Anhang 4.1.10	wallet_patch.py	328
Anhang 4.1.11	connection_target_patch.py.....	330
Anhang 4.1.12	validator_patch.py	333
Anhang 4.1.13	key_types.py	335
Anhang 4.1.14	key_type_patches.py	336
Anhang 4.1.15	multicodec_patch.py	339
Anhang 4.1.16	setup.py	341

Anhang 4.1.17	README.md	342
Anhang 4.2	Dockerfile: acapy-base-pqc	346
Anhang 4.3	docker-compose.yml: SSI Agenten mit acapy-base-pqc & Plugin	350
Anhang 4.4	Issuer Agent mit PQC-Plugin Boot Log	356
Anhang 5	Summative Evaluation	360
Anhang 5.1	KRITIS Szenario	360
Anhang 5.1.1	Teil 1: Setup & Verbindungstests	360
Anhang 5.1.2	Teil 2: DID Setup & Ledger Registration - KRITIS-Identitäten	370
Anhang 5.1.3	Teil 3: Schema & Credential Definition - Notfall-Wartungszertifikat ..	378
Anhang 5.1.4	Teil 4: Agent Connections - KRITIS-Akteure	394
Anhang 5.1.5	Teil 5: Credential Issuance - Notfall-Wartungszertifikat	419
Anhang 5.1.6	Teil 6: Proof Presentation - Zutrittskontrolle am Umspannwerk	425
Anhang 5.1.7	Teil 7: Revocation & Deletion- Beendigung der Wartungsberechtigung	438
Anhang 5.2	Validierung der KRITIS-Compliance-Anforderungen	459
Anhang 5.3	Validierung der Kryptoagilität - Applikationslayer	461

Anhang 1: Self-Sovereign Identity

Tabelle A-1
Zehn Prinzipien von Self-Sovereign Identity

Prinzip	Beschreibung
Existenz	Benutzer müssen eine unabhängige Existenz haben
Kontrolle	Benutzer müssen die Kontrolle über ihre Identität behalten
Zugriff	Benutzer müssen Zugriff auf ihre eigenen Daten haben
Transparenz	Systeme und Algorithmen müssen transparent sein
Persistenz	Identitäten müssen langlebig sein
Übertragbarkeit	Informationen und Dienste rund um die Identität müssen übertragbar sein
Interoperabilität	Identitäten sollten so weit wie möglich nutzbar sein
Einverständnis	Die Nutzer müssen der Verwendung ihrer Identität zustimmen
Minimalisierung	Die Offenlegung von Daten muss auf ein Minimum beschränkt werden
Schutz	Die Rechte der Nutzer müssen geschützt werden

Anmerkung. Basierend auf Allen (2016).

Anhang 2: Systematische Literaturrecherche

Die systematische Literaturrecherche bildet das wissenschaftliche Fundament dieser Masterarbeit und wurde in zwei methodisch konsistenten Iterationen durchgeführt, um eine höchstmögliche Aktualität im dynamischen Forschungsfeld der PQC und SSI zu gewährleisten.

Die erste Iteration fand im Rahmen des Exposés statt und etablierte die methodische Grundlage für das gesamte Forschungsprojekt. Diese initiale Recherche identifizierte 61 relevante Quellen mit differenzierter Relevanzklassifizierung (hoch/mittel/niedrig) und ermöglichte die Formulierung der Forschungsfragen sowie die Identifikation der Forschungslücke im Bereich PQC für SSI-Systeme in KRITIS. Im Ergebnis dieser ersten Phase konnten fünf Kernpublikationen identifiziert werden, die die Basis der theoretischen Fundierung bildeten. Die detaillierte Dokumentation dieser ersten Iteration ist in Anhang 2.1 integriert, um Transparenz und Reproduzierbarkeit des ursprünglichen Forschungskonzepts zu dokumentieren.

Die zweite Iteration (Anhang 2.2) wurde während der Ausarbeitungsphase der Masterarbeit durchgeführt, um den Zeitraum zwischen dem Exposé und dem Abschluss der Arbeit abzudecken (30. Mai 2025 bis 02. November 2025). Unter Anwendung der identischen Suchstrategie und Selektionskriterien nach PRISMA 2020 wurden in dieser Phase weitere 34 Quellen identifiziert.

Das kumulierte Ergebnis beider Iterationen umfasst somit die Identifikation und Analyse von insgesamt sieben hochrelevanten wissenschaftlichen Quellen, die den aktuellen Stand der Forschung in den Schnittmengen der Domänen SSI, Blockchain, PQC und KRITIS repräsentieren.

Anhang 2.1: Erste Iteration vom 30. Mai 2025 (Exposé)

Die systematische Literaturrecherche zum Stand der Forschung orientiert sich am „iterative Review-Ansatz“ nach Brocke et al. (2015, S. 208–209), der mit einer initialen Recherche startet und sich iterativ vertieft. Für die Struktur und Dokumentation sind ausgewählte Methoden der PRISMA 2020 Richtlinien zugrunde gelegt (Tabelle A-2). PRISMA gewährleistet hierbei einen transparenten, reproduzierbaren Prozess und verbessert die Berichtqualität (Page et al., 2021, S. 1, 6).

Tabelle A-2
Ausgewählte Methoden der PRISMA 2020 Richtlinien

Methode	Beschreibung
Ein- und Ausschlusskriterien	Ein- und Ausschlusskriterien für die Überprüfung.
Suchstrategie	Vollständigen Suchstrategie für alle Datenbanken, Websites einschließlich aller verwendeten Filter.
Selektionsprozess	Methoden an, die verwendet wurden, um zu entscheiden, ob eine Studie die Einschlusskriterien der Überprüfung erfüllt.

Anmerkung. In Anlehnung an Page et al. (2021, S. 4).

Die in Tabelle A-3 dargestellten Ein- und Ausschlusskriterien gewährleisten eine transparente, nachvollziehbare und zielgerichtete Auswahl relevanter wissenschaftlicher Quellen.

Tabelle A-3

Ein- und Ausschlusskriterien für die systematische Literaturrecherche

Kategorie	Einschluss	Ausschluss	Begründung
Thematischer Fokus	SSI und dezentrale Identitätslösungen; Blockchain-basierte Identitätsmanagementsysteme; PQC und quantensichere Algorithmen; Sicherheit und Compliance in KRITIS; DSR-Methodik in IT/Informationssystemen; Kontext; Kryptoagilität und kryptografische Migration	Identitätsmanagement ohne Bezug zu SSI oder Blockchain; Klassische PKI ohne PQC-Bezug; Kryptografie ohne Post-Quantum-Relevanz; Arbeiten ohne Bezug zu KRITIS oder ohne sicherheitskritischen IT/Informationssystemen; Kontext; Nicht-DSR-basierte Entwicklungsansätze	Fokussierung auf die Forschungsfragen und relevante technologische, methodische und regulatorische Aspekte.
Zeitrahmen	2015 bis heute	Vor 2015	Berücksichtigung aktueller technologischer Entwicklungen (Blockchain, PQC, SSI) und regulatorischer Anforderungen.

Fortsetzung auf nächster Seite

Tabelle A-3 (Fortsetzung)

Kategorie	Einschluss	Ausschluss	Begründung
Publikationstypen	Peer-reviewed Journalartikel; Konferenzbeiträge anerkannter Fachgesellschaften (z. B. IEEE, ACM, IFIP); Preprints; Offizielle Standards und Empfehlungen (z. B. NIST, W3C, BSI); Whitepaper etablierter Organisationen; Dissertationen und anerkannte Fachbücher	Blogposts, Forenbeiträge, Marketingmaterial; Po- pularwissenschaftliche Artikel ohne wissenschaftliche Fundierung; Unveröffentlichte Manuskripte ohne Peer-Review; Seminar- und Abschlussarbeiten ohne wissenschaftliche Begutachtung	Sicherstellung wissenschaftli- cher Qualität, Nachvollziehbar- keit und Relevanz der Quellen für die Masterarbeit; Da das For- schungsthema aktuell noch sehr neu ist und die einschlägige Fachliteratur teilweise noch nicht den Peer- Review-Prozess durchlaufen hat, werden auch Preprints in die Analyse einbezogen. Preprints ermöglichen eine zeitnahe Verfügbarkeit aktueller Forschungser- gebnisse, was insbesondere bei diesem innovativen von großer Bedeutung ist.

Fortsetzung auf nächster Seite

Tabelle A-3 (Fortsetzung)

Kategorie	Einschluss	Ausschluss	Begründung
Sprache	Deutsch; Englisch	Andere Sprachen als Deutsch und Englisch	Gewährleistung der Verständlichkeit und Zugänglichkeit für den deutsch- und englischsprachigen Forschungskontext.
Zugänglichkeit	Verfügbare Volltexte	Nicht verfügbare Volltexte	Ermöglichung einer gründlichen Analyse und Bewertung der Inhalte.

Anmerkung. Eigene Darstellung.

Tabelle A-4 stellt ausgewählte methodische Schritte zur Entwicklung einer Suchstrategie nach Bramer et al. (2018, S. 532) dar, an denen sich diese Seminararbeit orientiert.

Tabelle A-4
Überblick über die Entwicklung der Suchstrategie

Schritt	Beschreibung
1	Identifikation relevanter Schlüsselkonzepte
2	Identifikation relevanter Keywords
3	Erstellung einer strukturierten Suchanfrage mit Booleschen Operatoren
4	Auswahl geeigneter Datenbanken
5	Übersetzung der Suchanfrage für verschiedene Datenbanken

Anmerkung. In Anlehnung an Bramer et al. (2018, S. 532).

Identifikation relevanter Schlüsselkonzepte

Im ersten Schritt wird eine präzise Identifikation und Abgrenzung zentraler Schlüsselkonzepte vorgenommen (Tabelle A-5).

Tabelle A-5
Abgrenzung zentraler Schlüsselkonzepte

Schlüsselkonzept	Erläuterung
Self-Sovereign Identity	Das Paradigma der selbstbestimmten digitalen Identität, das Nutzenden die Kontrolle über ihre Identitätsdaten und deren Weitergabe ermöglicht, basierend auf dezentralen Technologien wie Blockchain und interoperablen Standards wie DID ,VC und VP (Solavagione & Vesco, 2025, S. 2).
Blockchain-Technologie	Die Nutzung von DLT zur Sicherstellung von Integrität, Transparenz und Manipulationssicherheit im Identitätsmanagement, insbesondere im Kontext von SSI-Systemen (Solavagione & Vesco, 2025, S. 2).
Post-Quantum Kryptografie	Kryptografische Verfahren, die auch gegen Angriffe durch leistungsfähige Quantencomputer resistent sind, einschließlich aktueller Standardisierungsansätze (z. B. NIST FIPS 203, 204, 205) (National Institute of Standards and Technology, 2024a, 2024b, 2024c) und Empfehlungen zur kryptoagilen Systemgestaltung.
Kritische Infrastrukturen	Sektoren und Systeme, deren Funktionsfähigkeit essenziell für das Gemeinwesen ist und die daher besonders hohe Anforderungen an Sicherheit, Compliance und Resilienz stellen (Bundesministerium der Justiz, 2009).
Design Science Research	Die methodische Grundlage zur systematischen Entwicklung, Evaluation und wissenschaftlichen Fundierung innovativer IT-Artefakte nach Hevner et al. (2004) im Kontext der genannten Technologien und Anwendungsdomänen.

Anmerkung. Basierend auf Solavagione und Vesco (2025, S. 2), National Institute of Standards and Technology (2024a, 2024b, 2024c), Bundesministerium der Justiz (2009), Hevner et al. (2004).

Identifikation relevanter Keywords

Basierend auf den zuvor definierten Schlüsselkonzepten wurden die wichtigsten Suchbegriffe in Deutsch und Englisch sowie die gängigen Akronyme zusammengestellt (Tabelle A-6), um die technologische, methodische und regulatorische Breite der Recherche optimal abzudecken.

Tabelle A-6

Keywords der Schlüsselkonzepte

Schlüsselkonzept	Keywords
Self-Sovereign Identity	Self-Sovereign Identity, SSI, dezentrale Identität, digitale Identität, decentralized identity, user-controlled identity, identity wallet, Verifiable Credentials, VC, Decentralized Identifiers, DID, Identity Management, IdM, Identity Access Management, IAM
Blockchain-Technologie	Blockchain, Distributed Ledger Technology, DLT, Distributed Ledger, Smart Contract, Ethereum, Hyperledger, IOTA, Permissioned Ledger, Consensus Mechanism, On-Chain Identity
Post-Quantum Kryptografie	Post-Quantum Cryptography, PQC, quantensichere Verschlüsselung, quantum-resistant cryptography, Lattice-based Cryptography, Hash-based Signature, Code-based Cryptography, Multivariate Cryptography, Module-Lattice-based Digital Signature, ML-DSA, Module-Lattice-based Key Encapsulation Mechanism, ML-KEM, Stateless Hash-based Digital Signature, SLH-DSA, CRYSTALS-Dilithium, SPHINCS+, Kryptoagilität, Cryptographic Agility, Algorithm Agility, Migration Strategy, Cryptographic Migration, Flexible Cryptography Update
Kritische Infrastrukturen	Kritische Infrastrukturen, KRITIS, Critical Infrastructure Protection, CIP, sector-specific security requirements, Compliance, Privacy by Design, Privacy by Default, Data Protection, Regulatory Requirements, Bundesamt für Sicherheit in der Informationstechnik, BSI, European Union Agency for Cybersecurity, ENISA, eIDAS

Fortsetzung auf nächster Seite

Tabelle A-6 (Fortsetzung)

Schlüsselkonzept	Keywords
Design Science Research	Design Science Research, DSR, Design Science Research Methodology, DSRM, Artefact Development, Evaluation of Artefacts, Method-driven Development, Iterative Design Process, Research Methodology, Framework for Evaluation in Design Science, FEDS, Preferred Reporting Items for Systematic Reviews and Meta-Analyses, PRISMA

Anmerkung. Eigene Darstellung.

Erstellung einer strukturierten Suchanfrage mit Booleschen Operatoren

Listing A-1 stellt eine strukturierte Suchstrategie mit Booleschen Operatoren auf Basis der identifizierten Keywords dar. Die Suchanfrage verbindet zentrale Schlüsselkonzepte und nutzt gezielt Synonyme und Akronyme zur Abdeckung verschiedener Terminologien und Schreibweisen.

Listing A-1

Strukturierte Suchanfrage mit Booleschen Operatoren

```

1 (
2 (
3 "self-sovereign identity" OR SSI OR "decentralized identity" OR
   "dezentrale Identität" OR "digitale Identität" OR "user-
   controlled identity" OR "identity wallet" OR "verifiable
   credentials" OR VC OR "decentralized identifiers" OR DID OR
   "identity management" OR IdM OR "identity access management"
   OR IAM
4 )
5 AND
6 (
7 blockchain OR "distributed ledger technology" OR DLT OR "
   distributed ledger" OR "smart contract" OR Ethereum OR
   Hyperledger OR IOTA OR "permissioned ledger" OR "consensus
   mechanism" OR "on-chain identity"
8 )
9 AND

```

```

10 (
11 "post-quantum cryptography" OR PQC OR "quantensichere Verschlü-
sselung" OR "quantum-resistant cryptography" OR "lattice-
based cryptography" OR "hash-based signature" OR "code-based
cryptography" OR "multivariate cryptography" OR "module-
lattice-based digital signature" OR ML-DSA OR "module-
lattice-based key encapsulation mechanism" OR ML-KEM OR "
stateless hash-based digital signature" OR SLH-DSA OR
CRYSTALS-Dilithium OR SPHINCS+ OR kryptoagilität OR "
cryptographic agility" OR "algorithm agility" OR "migration
strategy" OR "cryptographic migration" OR "flexible
cryptography update"
12 )
13 AND
14 (
15 "kritische Infrastrukturen" OR KRITIS OR "critical
infrastructure protection" OR CIP OR "sector-specific
security requirements" OR compliance OR "privacy by design"
OR "privacy by default" OR "data protection" OR "regulatory
requirements" OR "bundesamt für sicherheit in der
informationstechnik" OR BSI OR "european union agency for
cybersecurity" OR ENISA OR eIDAS
16 )
17 AND
18 (
19 "design science research" OR DSR OR "design science research
methodology" OR DSRM OR "artefact development" OR "
evaluation of artefacts" OR "method-driven development" OR "
iterative design process" OR "research methodology" OR "
framework for evaluation in design science" OR FEDS OR "
preferred reporting items for systematic reviews and meta-
analyses" OR PRISMA
20 )
21 )

```

Auswahl einer geeigneten Datenbank

Die Wahl auf EBSCO als Datenbank für die systematische Literaturrecherche resultiert aus der Existenz von Campuslizenzen der FOM für diese Datenbank.

Übersetzung der Suchanfrage für EBSCO

Die strukturierten Suchanfrage mit den Booleschen Operatoren kann für EBSCO ohne Anpassungen übernommen werden. Das Ergebnis der EBSCO Suchanfrage ist in Abbildung A-1 dargestellt. Insgesamt wurden mit dieser Abfrage 61 Quellen identifiziert.

Abbildung A-1

Erste Iteration - Ergebnis der EBSCO Suchanfrage

The screenshot shows the EBSCO search results interface. At the top, there is a search bar with the query: ("self-sovereign identity" OR SSI) OR ("decentralized identity" OR "digitale Identität" OR "user-controlled identity") OR ("identity wallet" OR "verifiable credentials"). Below the search bar, there are several filter options: Alle Filter (3), Verfügbare Volltexte, Peer-Reviewed, and QuellenTyp. The results section displays two articles:

- Survey of Blockchain-Based Applications for IoT.** By Enaya Ahmad, Fernando Xavier, Kashif Rasha. In: Applied Sciences (2026-3417), Apr 2025 • Complementary Index. This article is peer-reviewed and published in a journal. It discusses the rapid growth of the Internet of Things (IoT) and its challenges related to security, scalability, and data integrity. It explores blockchain technology's potential to address these issues.
- Securing Decentralized Ecosystems: A Comprehensive Systematic Review of Blockchain Vulnerabilities, Attacks, and Countermeasures and Mitigation Strategies.** By Siam, Md Kamruj; Saha, Blash; Hasan, Md Mehedi. In: Future Internet, Apr 2025 • Complementary Index. This article is peer-reviewed and published in a journal. It provides a systematic review of blockchain vulnerabilities across various industries, including healthcare and finance. It examines existing countermeasures and mitigation strategies, highlighting the need for context-aware security designs.

At the bottom left, the FOM Hochschule logo is visible. On the right side, there is a sidebar with links: Mein Dashboard, Neue Suche, Publication Finder, Concept-Map, Zusätzliche Quellen, Hilfe, Datenbankkatalog, and Links. A note at the bottom right says: Zugang zum bisherigen Ordner.

Anmerkung. Eigene Darstellung.

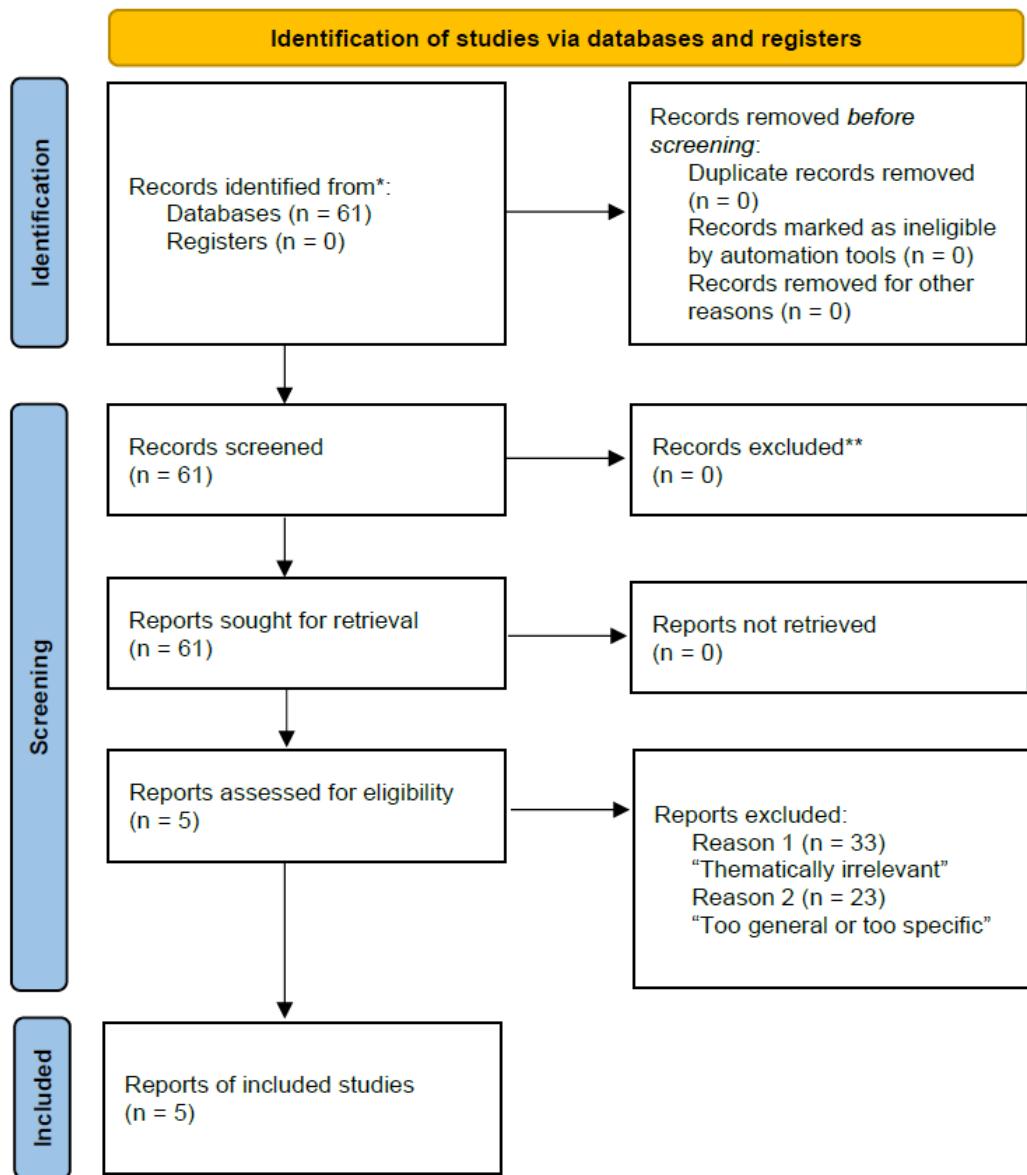
Selektionsprozess

Im nächsten Schritt folgt der Selektionsprozess der 61 identifizierten Quellen aufbauend auf den zuvor definierten Schlüsselkonzepten und der Suchstrategie. Dabei werden die in Tabelle A-3 definierten Ein- und Ausschlusskriterien konsequent angewendet, sodass nur fachlich einschlägige, qualitativ belastbare und für die Forschungsfragen zentrale Studien inkludiert werden.

Zunächst werden alle über die EBSCO-Suchanfrage identifizierten 61 Treffer einer Duplikatsbereinigung unterzogen und anschließend im Rahmen des Titel- und Abstract-Screenings hinsichtlich ihres thematischen Fokus bewertet. Hohe Relevanz erhalten Quellen mit klaren Beiträgen zu SSI, PQC, KRITIS oder dezentralen Identitätsarchitekturen, während Arbeiten zu angrenzenden Technologien wie Blockchain-Sicherheit im IoT oder digitaler Forensik als mittel relevant und allgemeine Technologietrends ohne direkten Bezug zum Thema als niedrig relevant eingestuft werden.

Abbildung A-2 fasst diesen Selektionsprozess der ersten Iteration zusammen. Nach dem Screening aller 61 Publikationen folgte die Eignungsprüfung, in deren Ergebnis fünf Publikationen die Einschlusskriterien erfüllten und 56 ausgeschieden wurden, da sie entweder thematisch irrelevant ($n = 33$) oder hinsichtlich ihrer Spezifität unangemessen waren ($n = 23$).

Abbildung A-2
Erste Iteration - PRISMA 2020 Flussdiagramm



Anmerkung. In Anlehnung an Page et al. (2021, S. 5).

Tabelle A-7 stellt eine Übersicht der Bewertung der 61 identifizierten Quellen dar, welche vollständig in Tabelle A-8 aufzufinden ist.

Tabelle A-7

Erste Iteration - Übersicht der relevanten Quellen

Nr.	Quelle	Relevanz
1	Szymanski, T. H. (2024). A Quantum-Safe Software-Defined Deterministic Internet of Things (IoT) with Hardware-Enforced Cyber-Security for Critical Infrastructures. <i>Information</i> (2078-2489), 15(4), 173. https://doi.org/10.3390/info15040173	Hoch
2	Nouma, S. E., & Yavuz, A. A. (2024). Trustworthy and Efficient Digital Twins in Post-Quantum Era with Hybrid Hardware-Assisted Signatures. <i>ACM Transactions on Multimedia Computing, Communications & Applications</i> , 20(6), 1–30. https://doi.org/10.1145/3638250	Hoch
3	Sharif, A., Ranzi, M., Carbone, R., Sciarretta, G., Marino, F. A., & Ranise, S. (2022). The eIDAS Regulation: A Survey of Technological Trends for European Electronic Identity Schemes. <i>Applied Sciences</i> (2076-3417), 12(24), 12679. https://doi.org/10.3390/app122412679	Hoch
4	Alam, M., Hoffstein, J., & Cambou, B. (2024). Privately Generated Key Pairs for Post Quantum Cryptography in a Distributed Network. <i>Applied Sciences</i> (2076-3417), 14(19), 8863. https://doi.org/10.3390/app14198863	Hoch
5	Radanliev, P. (2023). Review and Comparison of US, EU, and UK Regulations on Cyber Risk/Security of the Current Blockchain Technologies: Viewpoint from 2023. <i>Review of Socionetwork Strategies</i> , 17(2), 105–129. https://doi.org/10.1007/s12626-023-00139-x	Hoch
6–38	Diverse	Mittel
39–61	Diverse	Niedrig

Anmerkung. Basierend auf Tabelle A-8 und Titel und Abstracts von Alam, Hoffstein und Cambou (2024), Nouma und Yavuz (2024), Radanliev (2023), Sharif et al. (2022) und Szymanski (2024).

Bewertung der identifizierten Quellen hinsichtlich ihrer Relevanz

Tabelle A-8

Erste Iteration - Relevanzbewertung der identifizierten Quellen

Nr.	Quelle	Relevanz	Kommentar
1	Szymanski, T. H. (2024). A Quantum-Safe Software-Defined Deterministic Internet of Things (IoT) with Hardware-Enforced Cyber-Security for Critical Infrastructures. Information (2078-2489), 15(4), 173. https://doi.org/10.3390/info15040173	Hoch	Fokussiert auf die Entwicklung quantensicherer Kommunikations- und Sicherheitssysteme im Kontext von KRITIS und Industrial IoT; adressiert explizit PQC durch Einsatz quantensicherer Verschlüsselungsmechanismen und QKD-Netzwerke; behandelt hardwarebasierte Zugriffskontrollen, Zero Trust Architekturen und AI-gestützte Sicherheit, die für Resilienz und Sicherheitsanforderungen in KRITIS maßgeblich sind; zwar steht SSI und Blockchain-Technologie nicht im Mittelpunkt, jedoch zeigen die vorgestellten innovativen Konzepte und die experimentelle Validierung einen sehr hohen Anwendungs- und Erkenntniswert für den methodischen und technologischen Fortschritt in mindestens zwei zentralen Domänen (PQC, KRITIS); damit bietet der Beitrag substanzelle Impulse für den Schutz hochsensibler digitaler Infrastrukturen.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
2	Nouma, S. E., & Yavuz, A. A. (2024). Trustworthy and Efficient Digital Twins in Post-Quantum Era with Hybrid Hardware-Assisted Signatures. <i>ACM Transactions on Multimedia Computing, Communications & Applications</i> , 20(6), 1–30. https://doi.org/10.1145/3638250	Hoch	Betont die Notwendigkeit verlässlicher und effizienter digitaler Signaturen im Kontext von Digital Twins, die primär auf IoT-Infrastrukturen für hochsensible Daten abzielen und damit eine wichtige Schnittmenge zu KRITIS darstellen; adressiert explizit PQC durch die Entwicklung und Umsetzung quantensicherer und hybrider Signaturlösungen—einschließlich Forward Security und Aggregation, die auch für Blockchain-basierte und dezentrale Identitätsanwendungen (insb. mit Skalierungsbedarf) hoch relevant sind; der methodische Fortschritt im Bereich hardwareunterstützter, ressourcenschonender Kryptografie bietet substanzielle Innovationsimpulse für sicherheitskritische Systeme mit beschränkten Ressourcen, wie sie für SSI-Lösungen und die sichere Verwaltung digitaler Identitäten in KRITIS essenziell sind; Blockchain-Technologie und SSI werden nicht explizit vertieft, jedoch ist die Übertragbarkeit der vorgestellten Konzepte—insbesondere hybride und aggregierbare Signaturen—auf beide Domänen methodisch und praxisnah gegeben.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
3	Sharif, A., Ranzi, M., Carbone, R., Sciarretta, G., Marino, F. A., & Ranise, S. (2022). The eIDAS Regulation: A Survey of Technological Trends for European Electronic Identity Schemes. <i>Applied Sciences</i> (2076-3417), 12(24), 12679. https://doi.org/10.3390/app122412679	Hoch	Adressiert zentrale Entwicklungen und Herausforderungen europäischer elektronischer Identitätssysteme im Kontext der eIDAS-Regulierung; analysiert technologische Trends und ihre Auswirkungen auf Sicherheit, Datenschutz und Interoperabilität nationaler eID-Lösungen, was unmittelbar an die Domäne SSI und deren regulatorisches Umfeld anschließt; behandelt aktuelle Technologiestandards wie OAuth 2.0, SAML und OpenID Connect, ohne explizit Blockchain- oder PQC-Lösungen zu integrieren, beleuchtet jedoch die (in eIDAS 2.0 antizipierte) Entwicklung hin zu dezentralisierten Identitätsarchitekturen, die als Grundlage künftiger SSI-Lösungen dienen; liefert wesentliche Erkenntnisse für die Ausgestaltung sicherer und interoperabler digitaler Identitäten in KRITIS und gibt Impulse für die technologische und methodische Weiterentwicklung nationalübergreifender Identitätsverwaltung.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
4	Alam, M., Hoffstein, J., & Cambou, B. (2024). Privately Generated Key Pairs for Post Quantum Cryptography in a Distributed Network. <i>Applied Sciences</i> (2076-3417), 14(19), 8863. https://doi.org/10.3390/app14198863	Hoch	Fokussiert auf die praktische Erzeugung, Verteilung und Verifikation privat generierter post-quanten-sicherer Schlüsselpaare in verteilten Netzwerken, mit expliziter Anwendung von Crystals-Dilithium als PQC-Algorithmus; adressiert wesentlich die Domäne PQC durch Integration und Umsetzung eines aktuellen Standards, was sowohl für die Sicherheit verteilter Infrastrukturen als auch für zukünftige Identitätslösungen (z.B. im Kontext von SSI) zentral ist; Berücksichtigung von Multi-Faktor-Authentifizierung, Challenge-Response-Mechanismen und biometrielosen Verfahren bietet substanzIELLE methodische Impulse für die Entwicklung von sicheren, dezentralen Schlüsselmanagement- und Authentifizierungslösungen; direkte Einbindung in Blockchain- oder spezifische KRITIS wird nicht explizit diskutiert, ist aufgrund des Protokoll- und PKI-Fokus jedoch technisch anschlussfähig und für die Domänen SSI und KRITIS innovativ und relevant.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
5	Radanliev, P. (2023). Review and Comparison of US, EU, and UK Regulations on Cyber Risk/Security of the Current Blockchain Technologies: Viewpoint from 2023. <i>Review of Socionetwork Strategies</i> , 17(2), 105–129. https://doi.org/10.1007/s12626-023-00139-x	Hoch	Vergleichende Analyse der US-, EU- und UK-Regulierung im Bereich Cyber-Risiken und Sicherheit aktueller Blockchain-Technologien, basierend auf dem Stand 2023; systematische Prüfung und Gegenüberstellung führender Standardwerke wie NIST (US), ISO27001 (international), und neueren Regularien wie MiCA (EU) und CPMI-IOSCO, unter Einbeziehung technologieübergreifender Aspekte (u.a. PQC, Cloud Security, IoT); liefert bedeutende Einblicke und unmittelbaren Anwendungsbezug für die Bewertung, Weiterentwicklung und Integration internationaler Cybersecurity-Standards in neue Blockchain-Projekte—insbesondere mit Blick auf die vier Domänen moderner Identitäts- und Sicherheitssysteme
6	Enaya, A., Fernando, X., & Kashef, R. (2025). Survey of Blockchain-Based Applications for IoT. <i>Applied Sciences</i> (2076-3417), 15(8), 4562. https://doi.org/10.3390/app15084562	Mittel	Betrachtet zentrale Aspekte von Blockchain-Technologien und Sicherheit; Schwerpunkt auf IoT-Anwendungen und branchenspezifische Implementierungen; explizite Bezüge zu SSI, PQC und KRITIS fehlen; breiter Überblick, jedoch geringere Spezifität hinsichtlich der vier Domänen der Masterarbeit (SSI, Blockchain, PQC, KRITIS).

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
7	Siam, M. K., Saha, B., Hasan, M. M., Hossain Faruk, M. J., Anjum, N., Tahora, S., Siddika, A., & Shahriar, H. (2025). Securing Decentralized Ecosystems: A Comprehensive Systematic Review of Blockchain Vulnerabilities, Attacks, and Countermeasures and Mitigation Strategies. Future Internet, 17(4), 183. https://doi.org/10.3390/fi17040183	Mittel	Systematische Analyse von Schwachstellen, Angriffsszenarien und Gegenmaßnahmen in Blockchain-Ökosystemen; konzentriert sich auf Sicherheitsaspekte von Blockchain-Technologien ohne spezifische Betrachtung von SSI, PQC oder KRITIS; hoher inhaltlicher Wert für das allgemeine Verständnis von Blockchain-Sicherheit, jedoch eingeschränkte Anwendbarkeit auf alle vier Domänen der Masterarbeit.
8	Ramirez Lopez, L. J., & Morillo Ledezma, G. G. (2025). Employing Blockchain, NFTs, and Digital Certificates for Unparalleled Authenticity and Data Protection in Source Code: A Systematic Review. Computers (2073-431X), 14(4), 131. https://doi.org/10.3390/computers14040131	Mittel	Fokussiert auf Blockchain-basierte Technologien zur Sicherung von Authentizität und Zugriffskontrolle, jedoch im Anwendungskontext akademischer Quellcode-Sicherheit; behandelt primär NFTs und digitale Zertifikate, mit begrenztem Bezug zu SSI und ohne Einbeziehung von PQC; adressiert die Domäne „Blockchain“ und Aspekte der Datensicherheit, weist jedoch eine geringe Relevanz für die Domänen SSI, PQC und KRITIS im Kontext der Masterarbeit auf.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
9	Sebestyen, H., Popescu, D. E., & Zmaranda, R. D. (2025). A Literature Review on Security in the Internet of Things: Identifying and Analysing Critical Categories. <i>Computers</i> (2073-431X), 14(2), 61. https://doi.org/10.3390/computers14020061	Mittel	Umfassende Betrachtung aktueller Sicherheitsthemen und Identitätsmanagement im IoT-Kontext, unter Rückgriff auf neue Technologien wie Blockchain; Integrationspotenzial hinsichtlich Blockchain erkennbar, jedoch keine explizite Behandlung von SSI, PQC oder KRITIS; bietet einen breiten Überblick zu technologischen Lösungen und Herausforderungen, adressiert jedoch nur partiell die Anforderungen und Innovationspotenziale der vier Domänen der Masterarbeit.
10	Nambundo, J. M., de Souza Martins Gomes, O., de Souza, A. D., & Machado, R. C. S. (2025). Cybersecurity and Major Cyber Threats of Smart Meters: A Systematic Mapping Review. <i>Energies</i> (19961073), 18(6), 1445. https://doi.org/10.3390/en18061445	Mittel	Konzentriert sich auf Cybersecurity-Bedrohungen und Schwachstellen im Kontext von Smart Metern als Teil kritischer Infrastruktur; adressiert relevante Sicherheitsfragen in Bezug auf Energieversorgung, thematisch verwandt mit der Schutzbedarfsanalyse für KRITIS; der Einsatz von SSI, Blockchain-Technologien oder PQC wird nicht explizit thematisiert; bietet wichtige Einblicke in Bedrohungsszenarien und Mitigationsstrategien für smarte Energiesysteme, bleibt jedoch im Hinblick auf innovative Identitäts- oder Kryptografielösungen und deren methodischer Integration in Smart Metering-Systeme unspezifisch.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
11	Yuan, F., Huang, X., Zheng, L., Wang, L., Wang, Y., Yan, X., Gu, S., & Peng, Y. (2025). The Evolution and Optimization Strategies of a PBFT Consensus Algorithm for Consortium Blockchains. <i>Information</i> (2078-2489), 16(4), 268. https://doi.org/10.3390/info16040268	Mittel	Fokussiert auf die Optimierung des PBFT-Konsensalgorithmus, was grundlegende Bedeutung für Leistung, Sicherheit und Zuverlässigkeit von Consortium Blockchains hat; leistet methodischen Beitrag zur technologischen Weiterentwicklung im Blockchain-Bereich, jedoch ohne explizite Einbindung von SSI, PQC oder direktem Bezug zu KRITIS; relevante Erkenntnisse zur Verbesserung von Konsensmechanismen, die potenziell für skalierbare und sichere Blockchain-basierte Infrastrukturen adaptierbar sind, aber inhaltlich primär auf Konsensalgorithmen begrenzt.
12	Radanliev, P. (2024). Digital security by design. <i>Security Journal</i> , 37(4), 1640–1679. https://doi.org/10.1057/s41284-024-00435-3	Mittel	Bietet eine umfassende, technologieübergreifende Analyse aktueller Herausforderungen im Bereich digitale Sicherheit; adressiert relevante Zukunftsthemen wie AI, Blockchain und Quantencomputing im Kontext sich wandelnder Sicherheitsparadigmen; keine explizite Schwerpunktsetzung auf SSI, PQC oder spezifisch KRITIS; sektorübergreifende Betrachtung liefert wertvolle Einblicke zu regulatorischen und praxisbezogenen Aspekten, bleibt jedoch in Bezug auf die integrative Anwendung innovativer Sicherheitslösungen innerhalb der vier Domänen der Masterarbeit unspezifisch.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
13	Miller, T., Duriłk, I., Kostecka, E., Sokołowska, S., Kozłowska, P., & Zwolak, R. (2025). Artificial Intelligence in Maritime Cybersecurity: A Systematic Review of AI-Driven Threat Detection and Risk Mitigation Strategies. <i>Electronics</i> (2079-9292), 14(9), 1844. https://doi.org/10.3390/electronics14091844	Mittel	Fokus liegt auf der Anwendung von KI-gestützten Verfahren zur Erkennung und Minderung von Cyber-Bedrohungen im maritimen Sektor, adressiert damit relevante Aspekte kritischer Infrastrukturen; Schnittstellen zu Blockchain und quantenkryptographischen Ansätzen werden als Forschungsperspektiven genannt, ohne im Review zentrale methodische oder anwendungsbezogene Ausarbeitung zu bieten; SSI wird nicht behandelt, der primäre Schwerpunkt liegt auf KI und Cybersecurity, wodurch methodische und technische Details zu SSI und PQC im Kontext der vier Domänen der Masterarbeit fehlen.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
14	Atlam, H. F., Ekuri, N., Azad, M. A., & Lallie, H. S. (2024). Blockchain Forensics: A Systematic Literature Review of Techniques, Applications, Challenges, and Future Directions. <i>Electronics</i> (2079-9292), 13(17), 3568. https://doi.org/10.3390/electronics13173568	Mittel	Umfassende Analyse von Blockchain-Technologien im digitalen Forensik-Kontext mit Schwerpunkt auf Untersuchungsmethodik und Anwendungsfeldern; adressiert insbesondere die Herausforderungen beim Nachweis und der Verfolgung von Aktivitäten auf Blockchain-Systemen und beleuchtet regulatorische Aspekte, jedoch ohne explizite Behandlung von SSI, PQC oder den speziellen Anforderungen kritischer Infrastrukturen; liefert wertvolle Einblicke in forensische Anwendungen der Blockchain, bleibt jedoch mit Blick auf innovative Identitäts- und Kryptografielösungen sowie den Schutz kritischer Infrastrukturen im Rahmen der Masterarbeit begrenzt anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
15	Yakubu, M. M., Fadzil B Hassan, M., Danyaro, K. U., Junejo, A. Z., Siraj, M., Yahaya, S., Adamu, S., & Abdulsalam, K. (2024). A Systematic Literature Review on Blockchain Consensus Mechanisms' Security: Applications and Open Challenges. <i>Computer Systems Science & Engineering</i> , 48(6), 1437–1481. https://doi.org/10.32604/csse.2024.054556	Mittel	Umfassende systematische Analyse der Sicherheitsaspekte und Herausforderungen von Blockchain-Konsensmechanismen mit Fokus auf deren Integrität, Zuverlässigkeit und praktische Anwendungen; adressiert die Domäne „Blockchain“ in methodischer Tiefe und liefert wertvolle Erkenntnisse hinsichtlich Sicherheit, Skalierbarkeit und Energieeffizienz von Konsensprotokollen; explizite Bezüge zu SSI, PQC und spezifisch KRITIS fehlen, sodass die Übertragbarkeit auf die weiteren drei Domänen des Masterarbeit-Themas begrenzt ist.
16	Oude Roelink, B., El, H. M., & Sarmah, D. (2024). Systematic review: Comparing zk-SNARK, zk-STARK, and bulletproof protocols for privacy-preserving authentication. <i>Security & Privacy</i> , 7(5), 1–59. https://doi.org/10.1002/spy2.401	Mittel	Fokussiert auf den Vergleich und die Analyse moderner Zero-Knowledge-Protokolle (zk-SNARKs, zk-STARKs, Bulletproofs) mit hoher Relevanz für die Domänen Privacy und Blockchain, insbesondere im Kontext von Authentifizierung und Datenschutz; liefert methodische und performancebezogene Einblicke, jedoch ohne explizite Berücksichtigung von SSI oder PQC; Anbindung an KRITIS nicht direkt gegeben, bietet jedoch Potenzial für Integration innovativer Datenschutztechnologien in Blockchain-basierte Identitäts- und Authentifizierungssysteme.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
17	Cherbal, S., Zier, A., Hebal, S., Louail, L., & Annane, B. (2024). Security in internet of things: a review on approaches based on blockchain, machine learning, cryptography, and quantum computing. <i>Journal of Supercomputing</i> , 80(3), 3738–3816. https://doi.org/10.1007/s11227-023-05616-2	Mittel	Umfassende Analyse sicherheitsrelevanter Technologien im IoT-Kontext, darunter Blockchain, Kryptografie und Quantencomputing, mit breitem Überblick zu Ansätzen und Herausforderungen; adressiert insbesondere die Domänen Blockchain und PQC durch die Einbeziehung quantenkryptographischer und klassischer kryptographischer Lösungen, ohne explizite Behandlung von SSI oder dem spezifischen Anwendungsfeld kritischer Infrastrukturen; liefert wertvolle Vergleiche und Taxonomien zu modernen Sicherheitsmechanismen im IoT, bleibt jedoch hinsichtlich der methodischen und domänenspezifischen Vertiefung für SSI und KRITIS begrenzt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
18	Jagaramudi, G. K., Yazdinejad, A., Parizi, R. M., & Pouriyeh, S. (2024). Exploring privacy measurement in federated learning. <i>Journal of Supercomputing</i> , 80(8), 10511–10551. https://doi.org/10.1007/s11227-023-05846-4	Mittel	Fokussiert auf die Analyse von Privacy-Maßnahmen und deren Messbarkeit im Kontext von föderiertem Lernen, mit hohem methodischen Wert zur Bewertung von Datenschutz und Sicherheitsmetriken; Bezüge zu Blockchain oder SSI werden nicht explizit hergestellt, und PQC ist nur als Zukunftsperspektive am Rand angedeutet; die behandelten Konzepte und resultierenden Erkenntnisse zu Privacy-Measurement-Methoden können für sichere, dezentrale Systeme (einschließlich kritischer Infrastrukturen) grundsätzlich relevant sein, liefern jedoch keine spezifische oder anwendungsbezogene Vertiefung in den vier Kernbereichen der Masterarbeit.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
19	Alzoubi, Y. I., Gill, A., & Mishra, A. (2022). A systematic review of the purposes of Blockchain and fog computing integration: classification and open issues. <i>Journal of Cloud Computing</i> (2192-113X), 11(1), 1–36. https://doi.org/10.1186/s13677-022-00353-y	Mittel	Systematische Analyse der Integration von Blockchain-Technologien mit Fog Computing zur Lösung sicherheitsrelevanter Herausforderungen in IoT-Anwendungen; adressiert zentrale Aspekte wie Sicherheit, Datenschutz, Zugriffs- und Vertrauensmanagement, jedoch ohne explizite Behandlung von SSI oder konkreten Umsetzungen von PQC; verweist auf bestehende regulatorische und technologische Herausforderungen durch aufkommende Technologien wie Quantencomputing, aber ohne methodische oder anwendungsbezogene Vertiefung in Bezug auf KRITIS oder innovative Identitätskonzepte; bietet wertvolle Klassifikation und Überblick zu Blockchain-Anwendungen im Kontext verteilter Edge-Infrastrukturen, bleibt jedoch bezüglich der vier Kernbereiche der Masterarbeit auf die Domäne „Blockchain“ und allgemeine Sicherheitsaspekte beschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
20	Kazmi, S. H. A., Hassan, R., Qamar, F., Nisar, K., & Ibrahim, A. A. (2023). Security Concepts in Emerging 6G Communication: Threats, Countermeasures, Authentication Techniques and Research Directions. <i>Symmetry</i> (20738994), 15(6), 1147. https://doi.org/10.3390/sym15061147	Mittel	Umfassende Analyse sicherheitsrelevanter Konzepte, Bedrohungen und Authentifizierungsmethoden im Kontext der aufkommenden 6G-Kommunikation; adressiert innovative Technologien wie Künstliche Intelligenz, Quantencomputing und Föderiertes Lernen, wodurch potenzielle Schnittstellen zu PQC und Sicherheitsanforderungen für KRITIS bestehen; explizite Bezüge zu SSI und Blockchain-Technologien fehlen, ebenso eine methodische Vertiefung für spezielle SSI- oder Blockchain-basierte Authentifizierungslösungen; bietet wertvolle Einblicke in zukünftige Forschungsrichtungen und technologische Herausforderungen, jedoch begrenzte direkte Anwendbarkeit auf alle vier Kernbereiche der Masterarbeit.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
21	Attkan, A., & Ranga, V. (2022). Cyber-physical security for IoT networks: a comprehensive review on traditional, blockchain and artificial intelligence based key-security. <i>Complex & Intelligent Systems</i> , 8(4), 3559–3591. https://doi.org/10.1007/s40747-022-00667-z	Mittel	Fokussiert auf Authentifizierung und Schlüsselmanagement in IoT-Netzen unter Einbeziehung klassischer, Blockchain-basierter und KI-gestützter Sicherheitsmechanismen; adressiert im Wesentlichen die Domäne „Blockchain“ und bietet innovative Perspektiven zur dezentralen Verwaltung von Session-Keys sowie zu KI-basierten Angriffserkennungsmethoden im IoT-Kontext; SSI und PQC werden nicht explizit behandelt, ebenso fehlt die gezielte Anwendung auf KRITIS; der umfassende Überblick zu Authentifizierung und Schlüsselmanagement bietet methodische Anschlussmöglichkeiten für SSI-basierte Systeme oder kritische Infrastruktur, bleibt im Kern jedoch breit und technologieorientiert ohne vertiefte Ausarbeitung der vier Domänen der Masterarbeit.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
22	Ray, P. P. (2023). Web3: A comprehensive review on background, technologies, applications, zero-trust architectures, challenges and future directions. <i>Internet of Things & Cyber Physical Systems</i> , 3, 213–248. https://doi.org/10.1016/j.iotcps.2023.05.003	Mittel	Bietet einen breiten Überblick über die technologischen und gesellschaftlichen Grundlagen sowie Anwendungsfelder von Web3 und dezentralen Plattformen, wobei die Domäne Blockchain als zentrales Element systematisch behandelt wird; explizite Bezüge zu SSI und Zero-Trust-Architekturen existieren, jedoch fehlt eine detaillierte methodische Analyse spezifischer SSI-Lösungen oder Implementierungen, und PQC wird nicht thematisiert; der Beitrag verweist auf innovative Identitätskonzepte, Anwendungsintegration und technische Herausforderungen, bleibt jedoch hinsichtlich der anwendungsbezogenen Vertiefung zu PQC sowie spezifischen Schutzmaßnahmen für KRITIS auf einer konzeptionellen Ebene.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
23	Stach, C., Gritti, C., Bräcker, J., Behringer, M., & Mitschang, B. (2022). Protecting Sensitive Data in the Information Age: State of the Art and Future Prospects. Future Internet, 14(11), 302. https://doi.org/10.3390/fi14110302	Mittel	Umfassende Analyse aktueller Privacy-Mechanismen im Kontext datengetriebener Smart Services, mit Fokus auf die praktische Umsetzung nutzerfreundlicher Datenschutzlösungen; adressiert zentrale Herausforderungen und praktische Einsatzfelder moderner Datenschutzverfahren, ohne jedoch explizit auf SSI, Blockchain-Technologien, PQC oder spezielle Schutzanforderungen kritischer Infrastrukturen einzugehen; liefert wertvolle Einblicke zu datenorientierten Schutzmechanismen und deren Limitationen, bleibt jedoch hinsichtlich der methodischen und domänenspezifischen Vertiefung für SSI, PQC und KRITIS konzeptionell und generisch.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
24	<p>Farooq, M. S., Riaz, S., & Alvi, A. (2023). Security and Privacy Issues in Software-Defined Networking (SDN): A Systematic Literature Review. <i>Electronics</i> (2079-9292), 12(14), 3077. https://doi.org/10.3390/electronics12143077</p>	Mittel	<p>Systematische Analyse von Sicherheits- und Datenschutzproblemen in Software-Defined Networks mit Schwerpunkt auf Schwachstellen, Angriffen und Sicherungsmechanismen entlang der verschiedenen Netzwerkebenen; adressiert grundlegende Herausforderungen für den Schutz moderner Netzwerkarchitekturen, insbesondere durch die Trennung von Steuer- und Datenebene—ein Aspekt, der für den Betrieb kritischer Infrastrukturen relevante Einblicke und Methoden liefert; explizite Bezüge zu Blockchain-Technologien, SSI und PQC fehlen, jedoch kann die vorgestellte Taxonomie und die Diskussion zukünftiger Forschungsrichtungen methodische Impulse für sichere Integrationskonzepte in verteilten, kritischen oder identitätsgetriebenen Architekturen bieten—die inhaltliche Tiefe und Anwendbarkeit bleibt jedoch primär auf SDN-spezifische Herausforderungen fokussiert.</p>

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
25	Chanal, P. M., & Kakkasageri, M. S. (2020). Security and Privacy in IoT: A Survey. <i>Wireless Personal Communications</i> , 115(2), 1667–1693. https://doi.org/10.1007/s11277-020-07649-9	Mittel	Bietet einen umfassenden Überblick über grundlegende Sicherheits- und Datenschutzherausforderungen im Kontext des Internet of Things mit Fokus auf ressourcenbeschränkte Geräte; adressiert Kernaspekte wie Vertraulichkeit, Integrität, Authentifizierung und Verfügbarkeit, ohne jedoch explizit auf SSI, Blockchain-Technologien, PQC oder spezifische Anforderungen kritischer Infrastrukturen einzugehen; liefert wertvolle konzeptionelle Grundlagen zu Sicherheitsanforderungen und -architekturen im IoT, bleibt jedoch bezüglich anwendungs- oder methodenspezifischer Vertiefung zu den vier Kernbereichen der Masterarbeit generisch.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
26	Choudhary, A. (2024). Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions. <i>Discover Internet of Things</i> , 4(1), 1–41. https://doi.org/10.1007/s43926-024-00084-3	Mittel	Umfassende Übersicht und Analyse der IoT-Architektur, Anwendungen und Herausforderungen; adressiert technologische, soziale und funktionale Aspekte des Internet of Things, mit generischer Betrachtung von Architekturen und Simulationsumgebungen; explizite Bezüge zu SSI, Blockchain-Technologien, PQC oder spezifischen Schutzanforderungen kritischer Infrastrukturen fehlen; liefert wertvolle Grundlagen für das technologische Umfeld, bleibt jedoch hinsichtlich der vier Domänen der Masterarbeit konzeptionell und unspezifisch.
27	Sikiru, I. A., Kora, A. D., Ezin, E. C., Imoize, A. L., & Li, C.-T. (2024). Hybridization of Learning Techniques and Quantum Mechanism for IIoT Security: Applications, Challenges, and Prospects. <i>Electronics</i> (2079-9292), 13(21), 4153. https://doi.org/10.3390/electronics13214153	Mittel	Systematische Analyse hybrider Sicherheitsansätze in der Industrial IoT (IIoT), insbesondere durch die Kombination klassischer Lernverfahren und quantenmechanistischer Ansätze; adressiert relevante Herausforderungen und Perspektiven der IIoT-Sicherheit mit Berücksichtigung von Blockchain-Technologien und Quantum Mechanisms, wobei PQC eher implizit thematisiert wird; explizite Vertiefung von SSI und spezifische Anwendungen in KRITIS fehlen, liefert jedoch Impulse für die Integration moderner Kryptografie- und Sicherheitsverfahren im industriellen Umfeld.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
28	RadRadanliev, P. (2024). Artificial intelligence and quantum cryptography. <i>Journal of Analytical Science & Technology</i> , 14, 1–17. https://doi.org/10.1186/s40543-024-00416-6	Mittel	Thematisiert den aktuellen Stand und die Zukunftsperspektiven an der Schnittstelle von künstlicher Intelligenz und quantenkryptografischen Verfahren, wobei insbesondere der Einfluss von AI-Methoden auf Effizienz und Robustheit kryptografischer Systeme sowie die Herausforderungen durch das „Quantum Threat“-Szenario im Zentrum stehen; explizite Bezüge zu SSI, Blockchain-Technologien oder spezifischen Anwendungsfällen in KRITIS fehlen; liefert wertvolle Impulse zu methodischen Innovationen in der PQC, bleibt jedoch hinsichtlich der vier Domänen der Masterarbeit primär konzeptionell und technologisch fokussiert auf AI und Quantenkryptografie.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
29	O'Donoghue, O., Vazirani, A. A., Brindley, D., & Meinert, E. (2019). Design Choices and Trade-Offs in Health Care Blockchain Implementations: Systematic Review. <i>Journal of Medical Internet Research</i> , 21(5), e12426. https://doi.org/10.2196/12426	Mittel	Systematische Analyse von Architektur- und Design-Entscheidungen bei der Implementierung von Blockchain-Technologie im Kontext elektronischer Gesundheitsakte (EMR), mit Schwerpunkt auf sicherheitsrelevanten und skalierbaren Systemanforderungen sowie Trade-offs zwischen verschiedenen technischen, organisatorischen und anwendungsbezogenen Merkmalen; behandelt die Domäne Blockchain umfassend und liefert wertvolle Erkenntnisse über sicherheitsrelevante Kompromisse im Gesundheitswesen, adressiert jedoch weder SSI noch PQC oder KRITIS explizit; die Untersuchung des Spannungsfelds zwischen Sicherheit, Skalierbarkeit und Datenmanagement bildet eine methodisch relevante Grundlage, bleibt aber in Bezug auf die vier zentralen Domänen der Masterarbeit auf anwendungsbezogene Blockchain-Implementierungen beschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
30	Mulholland, J., Mosca, M., & Braun, J. (2017). The Day the Cryptography Dies. <i>IEEE Security & Privacy</i> , 15(4), 14–21. https://doi.org/10.1109/MSP.2017.3151325	Mittel	Überblickartige Darstellung der Auswirkungen von Quantencomputern auf bestehende kryptografische Verfahren; adressiert explizit die Bedrohung aktueller Sicherheitstechnologien (PQC), jedoch ohne methodische oder technologische Vertiefung zu Blockchain, SSI oder KRITIS; bietet konzeptionelle Einblicke in Risikoszenarien und Bedrohungsmodelle, bleibt jedoch hinsichtlich innovativer Lösungsansätze oder spezifischer Anwendungsgebiete der vier Domänen der Masterarbeit unspezifisch.
31	G, C. A., & Basarkod, P. I. (2024). A survey on blockchain security for electronic health record. <i>Multimedia Tools and Applications: An Journal</i> , 1–35. https://doi.org/10.1007/s11042-024-19883-5	Mittel	Fokus auf Blockchain-basierte Sicherheitslösungen für elektronische Gesundheitsakte (EHR) mit Einbindung von Deep-Learning-Methoden; adressiert primär die Domäne Blockchain durch Analyse von Datenschutz, Datensicherheit und Zugriffskontrolle im Gesundheitswesen, bietet wertvolle methodische Einblicke zur Anwendung verteilter Technologien im Bereich sensibler Daten; explizite Bezüge zu SSI, PQC und spezifisch KRITIS außerhalb des Gesundheitssektors fehlen, wodurch die Anwendbarkeit auf alle vier Domänen der Masterarbeit beschränkt bleibt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
32	Batta, P., Ahuja, S., & Kumar, A. (2024). Future Directions for Secure IoT Frameworks: Insights from Blockchain-Based Solutions: A Comprehensive Review and Future Analysis. <i>Wireless Personal Communications: An International Journal</i> , 139(3), 1749–1781. https://doi.org/10.1007/s11277-024-11694-z	Mittel	Systematische Untersuchung sicherer IoT-Frameworks unter Verwendung von Blockchain-Technologien; detaillierte Analyse verschiedener Algorithmen (u.a. Konsensmechanismen, RSA, Hashing) und Plattformen (Ethereum, CoSMOS, Hyperledger Fabric), mit Fokus auf die Verbesserung von Sicherheit und Performance in IoT-Systemen; explizite Bezüge zu SSI, PQC und den besonderen Anforderungen kritischer Infrastrukturen fehlen; adressiert vor allem die Domäne „Blockchain“ und liefert grundlegende Einblicke zur Anwendung verteilter Sicherheitsmechanismen im IoT, bleibt jedoch hinsichtlich der vier Kerndomänen der Masterarbeit (SSI, Blockchain, PQC, KRITIS) methodisch und domänen spezifisch eingeschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
33	Radanliev, P. (2024). Integrated cybersecurity for metaverse systems operating with artificial intelligence, blockchains, and cloud computing. <i>Frontiers in Blockchain</i> , 1–14. https://doi.org/10.3389/fbloc.2024.1359130	Mittel	Umfassende Analyse der Cybersicherheitslandschaft im Kontext integrierter Metaverse-Systeme unter Einbezug von Artificial Intelligence, Blockchain und Cloud Computing; adressiert zentrale Risikofelder, regulatorische Herausforderungen und die Rolle moderner Sicherheitstechnologien für die digitale Ökonomie, wobei insbesondere Blockchain in seiner Bedeutung für selbstverwaltete Systeme und Netzwerkgovernance diskutiert wird; explizite Vertiefungen zu SSI, PQC oder deren spezieller Anwendung im Umfeld kritischer Infrastrukturen fehlen, ebenso bleibt die methodische und technologische Anbindung an innovative Identitäts- und Kryptografieansätze im Rahmen der vier Domänen der Masterarbeit auf konzeptionelle Ausblicke beschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
34	Hajian Berenjestanaki, M., Barzegar, H. R., El Ioini, N., & Pahl, C. (2024). Blockchain-Based E-Voting Systems: A Technology Review. <i>Electronics</i> (2079-9292), 13(1), 17. https://doi.org/10.3390/electronics13010017	Mittel	Systematische Analyse von Blockchain-basierten E-Voting-Systemen mit Fokus auf Sicherheits-, Transparenz- und Integritätsaspekte; zentrale Bewertung technologischer Herausforderungen und zukünftiger Forschungsfragen, insbesondere zu Skalierbarkeit und Datenschutz – thematisch eng an die Domäne „Blockchain“ angelehnt; explizite Vertiefung von SSI, PQC oder die Anwendung in besonders schützenswerten KRITIS fehlt, bietet jedoch methodische Ansätze und technische Perspektiven, die für die Entwicklung sicherer und vertrauenswürdiger Abstimmungssysteme in digitalen Infrastrukturen relevant sein können.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
35	Pirbhulal, S., Chockalingam, S., Shukla, A., & Abie, H. (2024). IoT cybersecurity in 5G and beyond: a systematic literature review. <i>International Journal of Information Security</i> , 23(4), 2827–2879. https://doi.org/10.1007/s10207-024-00865-5	Mittel	Systematische Literaturübersicht zu Cybersicherheitsaspekten in 5G- und Next-Generation-IoT-Umgebungen, insbesondere hinsichtlich Threats, Authentifizierung, Zugriffskontrolle, Netzwerk- und Anwendungsschicht sowie Herausforderungen durch Softwarisierung und Virtualisierung der Netze; adressiert methodisch den aktuellen Forschungsstand, evaluiert genutzte Validierungsansätze (Praxis, Simulation, Theorie) und liefert ein Kategorienschema für existierende Sicherheitsmechanismen und offene Forschungsfragen; explizite Bezüge zu SSI, Blockchain-Technologien oder PQC fehlen, ebenso eine gezielte Betrachtung von KRITIS—die Branchenbeispiele (z. B. Healthcare, Energie) lassen eine indirekte Bedeutung für KRITIS erkennen, ohne diese jedoch methodisch zu vertiefen; methodische und technologische Tiefe für die vier Masterarbeitsdomänen beschränkt sich auf generische Cybersicherheitsbedrohungen und Lösungsansätze in modernen IoT/5G-Systemen.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
36	Asif, M., Abrar, M., Salam, A., Amin, F., Ullah, F., Shah, S., & AlSalman, H. (2025). Intelligent two-phase dual authentication framework for Internet of Medical Things. <i>Scientific Reports</i> , 15(1), 1–19. https://doi.org/10.1038/s41598-024-84713-5	Mittel	Fokus liegt auf der Entwicklung und Evaluierung eines intelligenten Zwei-Phasen-Authentifizierungsframeworks für die Internet of Medical Things (IoMT) mit Ziel der effizienten und sicheren Kommunikation sensibler Gesundheitsdaten; zentrale technische Ansätze umfassen ECDH für Schlüsselaustausch und AES-GCM für Datenverschlüsselung, wobei signifikante Verbesserungen in Effizienz und Sicherheit gegenüber klassischen Authentifizierungsmethoden nachgewiesen werden; explizite Bezüge zu SSI und Blockchain-Technologien sowie PQC fehlen vollständig, sodass methodische und technologische Innovationen in diesen Bereichen für den Rahmen der Masterarbeit unberücksichtigt bleiben; adressiert Schutzanforderungen im Bereich kritischer Infrastrukturen exemplarisch am Gesundheitswesen, bleibt jedoch in der Tiefe auf klassische kryptografische Verfahren und Authentifizierungsprozesse limitiert.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
37	Marengo, A., & Santamato, V. (2025). Quantum algorithms and complexity in healthcare applications: a systematic review with machine learning-optimized analysis. <i>Frontiers in Computer Science</i> , 1–30. https://doi.org/10.3389/fcomp.2025.1584114	Mittel	Systematische Übersicht zur Anwendung von Quantenalgorithmen und quanten-inspirierten Komplexitätsanalysen im Gesundheitswesen, mit zwei Schwerpunkten: (1) Quantum Computing für KI-basierte Analysen biomedizinischer Daten und (2) quantenkryptografische Protokolle zur Absicherung medizinischer Daten. Expliziter Bezug zur Domäne PQC durch Analyse quantensicherer und blockchain-basierter Sicherheitsmechanismen im Healthcare-Kontext. Keine explizite Behandlung von SSI oder dezierten Blockchain-Architekturen außerhalb sicherheitsrelevanter Frameworks; Anwendung auf KRITIS implizit durch den Fokus auf sichere medizinische Systeme, jedoch nicht technologieübergreifend vertieft. Insgesamt methodisch relevant für die Domäne PQC und für Sicherheitsthemen im medizinisch-kritischen Sektor, für die vier Themenbereiche der Masterarbeit aber primär im Bereich quantensicherer Daten- und KI-Anwendungen anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
38	Ahakonye, L. A. C., Nwakanma, C. I., & Kim, D.-S. (2024). Tides of Blockchain in IoT Cybersecurity. Sensors (14248220), 24(10), 3111. https://doi.org/10.3390/s24103111	Mittel	Umfassende Übersicht zu Anwendungsmöglichkeiten und Herausforderungen von Blockchain-Technologie im Bereich der IoT-Cybersicherheit, insbesondere in Verbindung mit KI-unterstützten Intrusion Detection Systemen; adressiert die Domäne „Blockchain“ grundlegend sowie deren Potenzial für Transparenz, Dezentralität und Unveränderlichkeit im IoT-Kontext; Integration von AI und Blockchain als Innovationstreiber für sichere und skalierbare IDS-Lösungen im IoT/IoT; SSI und PQC werden nicht explizit behandelt; spezifische Anforderungen und Anwendungsfälle für KRITIS werden nur indirekt adressiert; liefert wertvolle Einblicke und methodische Ansätze für die Weiterentwicklung sicherer IoT-Systeme, bleibt jedoch hinsichtlich der vier Domänen der Masterarbeit vorwiegend auf Blockchain und allgemeine Sicherheitsthemen im IoT fokussiert.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
39	Nain, A., Sheikh, S., Shahid, M., & Malik, R. (2024). Resource optimization in edge and SDN-based edge computing: a comprehensive study. <i>Cluster Computing</i> , 27(5), 5517–5545. https://doi.org/10.1007/s10586-023-04256-8	Niedrig	Umfassende systematische Analyse aktueller Optimierungsansätze für Ressourcenmanagement in Edge-Computing-Umgebungen, insbesondere unter Integration von Software-Defined Networking (SDN); adressiert zentrale Herausforderungen der effizienten Ressourcennutzung, Kontrollarchitekturen und Netzwerkprogrammierbarkeit, was insbesondere für leistungsfähige, latenzarme Anwendungen und Systemarchitekturen an der Netzwerkkante relevant ist; explizite Bezüge zu SSI, Blockchain-Technologien, PQC oder den besonderen Anforderungen kritischer Infrastrukturen fehlen; liefert dennoch wertvolle methodische Impulse für das Design verteilter, dynamischer Infrastrukturen, bleibt aber in Bezug auf die vier Domänen der Masterarbeit überwiegend allgemein und technologieorientiert.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
40	Netinant, P., Saengsuwan, N., Rukhiran, M., & Pukdesree, S. (2023). Enhancing Data Management Strategies with a Hybrid Layering Framework in Assessing Data Validation and High Availability Sustainability. Sustainability (2071-1050), 15(20), 15034. https://doi.org/10.3390/su152015034	Niedrig	Betrachtet Methoden zur nachhaltigen und hochverfügbaren Datenmigration, insbesondere durch ein hybrides Layering-Framework im Kontext von Data Management und Data Validation; adressiert primär Herausforderungen und Optimierungsstrategien im Bereich Datenkonsistenz, Datenintegrität und Verfügbarkeitsmanagement bei Migration und Transformation—relevant für unternehmensweite IT-Systeme und Logistikdaten; explizite Bezüge zu SSI, Blockchain-Technologien, PQC oder besonderen Anforderungen kritischer Infrastrukturen fehlen vollständig; liefert wertvolle Erkenntnisse zur Bewertung und Ausgestaltung von Datenmigrationsprozessen, bleibt jedoch hinsichtlich der vier Domänen der Masterarbeit methodisch und inhaltlich unberührt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
41	Trautman, L. J., Shackelford, S., Elzweig, B., & Ormerod, P. (2024). Understanding Cyber Risk: Unpacking and Responding to Cyber Threats Facing the Public and Private Sectors. University of Miami Law Review, 78(3), 840–916. https://repository.law.miami.edu/umlr/vol78/iss3/5/	Niedrig	Umfassende Analyse aktueller Cyberbedrohungen und deren Auswirkungen auf öffentliche und private Sektoren, mit Fokus auf Angriffsszenarien (u.a. Ransomware, Cyberwarfare, Datenlecks), regulatorische und unternehmensbezogene Steuerungsmechanismen sowie das Zusammenspiel von Recht, Unternehmensführung und geopolitischen Risiken; adressiert zentrale Aspekte der Cyber-Risikobewertung und Reaktion auf digitale Angriffe, insbesondere aus administrativer und juristischer Perspektive; explizite Bezüge zu SSI, Blockchain-Technologien, PQC sowie spezifische Schutzmaßnahmen für KRITIS fehlen; bietet wertvolle konzeptionelle Grundlagen im Bereich Cybersicherheit, Governance und Compliance, bleibt jedoch hinsichtlich der vier Schwerpunktgebiete der Masterarbeit in methodischer und technologischer Tiefe eingeschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
42	Hendaoui, F., Ferchichi, Niedrig A., Trabelsi, L., Meddeb, R., Ahmed, R., & Khelifi, M. K. (2024). Advances in deep learning intrusion detection over encrypted data with privacy preservation: a systematic review. <i>Cluster Computing</i> , 27(7), 8683–8724. https://doi.org/10.1007/s10586-024-04424-4	Niedrig	Systematische Analyse der Fortschritte im Bereich Deep-Learning-basierter Intrusion Detection über verschlüsselte Daten mit Fokus auf Privacy-Preservation; behandelt innovative Ansätze zur Anomalieerkennung in verschlüsselten Datenströmen durch tiefe neuronale Netze, ohne auf Datenentschlüsselung angewiesen zu sein; explizite Bezüge zu SSI, Blockchain-Technologien, PQC oder dem Schutz kritischer Infrastrukturen fehlen; bietet wertvolle methodische und technologische Impulse zur sicheren Datenverarbeitung und Angriffserkennung in datenschutzorientierten Systemen, bleibt aber hinsichtlich der vier Kerndomänen der Masterarbeit überwiegend auf den Bereich Deep Learning und Privacy-Preserving IDS fokussiert

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
43	Akartuna, E. A., Johnson, S. D., & Thornton, A. E. (2023). The money laundering and terrorist financing risks of new and disruptive technologies: a futures-oriented scoping review: The money laundering and terrorist financing risks of new and disruptive technologies: a futures-oriented scoping review. <i>Security Journal</i> , 36(4), 615–650. https://doi.org/10.1057/s41284-022-00356-z	Niedrig	Systematische Analyse von Geldwäsche- und Terrorismusfinanzierungsrisiken im Zusammenhang mit neuen und disruptiven Technologien, insbesondere Distributed-Ledger-Technologien (inkl. Kryptowährungen), neue Zahlungswege und FinTech; behandelt umfassend die Risiken, Methoden sowie betroffene Akteure und skizziert daraus resultierende Trends und politische Implikationen – mit klarem Bezug zur Domäne „Blockchain“ und angrenzender regulatorischer Herausforderungen; explizite Vertiefungen zu SSI oder PQC fehlen, ebenso eine gezielte Betrachtung kritischer Infrastrukturen im engeren technischen Sinne; liefert wichtige Einblicke für die Risiko- und Bedrohungsanalyse im Kontext innovativer Finanztechnologien, bleibt aber hinsichtlich der methodischen Tiefe und direkten Anwendbarkeit für die vier Domänen der Masterarbeit beschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
44	Zboril, M., & Svatá, V. (2025). Performance comparison of cloud virtual machines. <i>Journal of Systems & Information Technology</i> , 27(2), 197–213. https://doi.org/10.1108/JSIT-02-2022-0040	Niedrig	Thematischer Fokus liegt auf der vergleichenden Performancemessung von Cloud-basierten virtuellen Maschinen (VMs) bei AWS, Microsoft Azure und Google Cloud Platform, basierend auf Benchmark-Tests unter Linux; adressiert ausschließlich Infrastruktur- und Leistungsaspekte von Cloud-Diensten sowie Auswahlkriterien für IT-Betriebsmodelle; keine Verbindung zu den vier Domänen der Masterarbeit, da keine sicherheitsrelevanten, kryptografischen oder identitätsbezogenen Aspekte behandelt werden; relevante Erkenntnisse für Cloud-Infrastrukturmanagement und Benchmarking, jedoch methodisch und inhaltlich außerhalb des Kernbereichs der Masterarbeit.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
45	Radanliev, P. (2024). The rise and fall of cryptocurrencies: defining the economic and social values of blockchain technologies, assessing the opportunities, and defining the financial and cybersecurity risks of the Metaverse. Financial Innovation, 10, 1–34. https://doi.org/10.1186/ s40854-023-00537-8	Niedrig	Umfassende Analyse der wirtschaftlichen, sozialen und technologischen Aspekte von Blockchain-Technologien, insbesondere im Kontext von Kryptowährungen und deren Rolle im Metaverse; untersucht wirtschaftliche Chancen, Investitionsstrategien und Cybersecurity-Risiken mit interdisziplinärem Ansatz, inklusive Risikobewertung und maschinellem Lernen im Finanzsektor; explizite Bezüge zu SSI, PQC und dem Schutz kritischer Infrastrukturen fehlen, ebenso eine methodische oder technologische Vertiefung zu innovativen Identitäts- oder Kryptografielösungen—fokussiert primär auf ökonomische und anwendungsbezogene Fragestellungen der Blockchain im Finanz- und Metaverse-Umfeld.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
46	Bunescu, L., & Vârtei, A. M. (2024). Modern finance through quantum computing—A systematic literature review. PLoS ONE, 19(7), 1–22. https://doi.org/10.1371/journal.pone.0304317	Niedrig	Systematische Analyse des Einsatzes von Quantencomputing im Finanzsektor, mit Fokus auf Simulation, Optimierung und maschinelles Lernen; adressiert Kernaspekte der PQC im Hinblick auf die transformative Wirkung quantenbasierter Technologien, ohne dabei explizit auf Blockchain-Technologien oder SSI einzugehen; liefert wertvolle Einblicke in die methodische und anwendungsbezogene Entwicklung von Quantum Finance, bleibt jedoch hinsichtlich der vier thematischen Domänen der Masterarbeit (SSI, Blockchain, PQC, KRITIS) primär auf Finanzanwendungen und damit nur partiell anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
47	Alzoubi, Y. I., Mishra, A., & Topcu, A. E. (2024). Research trends in deep learning and machine learning for cloud computing security. <i>Artificial Intelligence Review: An International Science and Engineering Journal</i> , 57(5). https://doi.org/10.1007/s10462-024-10776-5	Niedrig	Fokussiert auf den Einsatz von Deep-Learning- und Machine-Learning-Technologien zur Identifikation und Bewältigung von Cloud-Sicherheitsbedrohungen; adressiert zentrale Herausforderungen wie Anomalieerkennung, Security Automation und die Integration neuer Technologien, ohne jedoch explizit SSI, Blockchain-Ansätze oder PQC systematisch einzubinden; hebt methodische, datenschutzbezogene und regulatorische Fragestellungen hervor, die für den Schutz kritischer Infrastrukturen relevant sind, bleibt jedoch bezüglich der vier Kernbereiche der Masterarbeit hauptsächlich auf Cloud Security und AI-gestützte Verfahren konzentriert und bietet nur indirekte Anschlussmöglichkeiten für innovative Kryptografie- oder Identitätslösungen

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
48	Williamson, S. M., & Prybutok, V. (2024). Balancing Privacy and Progress: A Review of Privacy Challenges, Systemic Oversight, and Patient Perceptions in AI-Driven Healthcare. <i>Applied Sciences</i> (2076-3417), 14(2), 675. https://doi.org/10.3390/app14020675	Niedrig	Kritische Analyse von Datenschutz-, Ethik- und Compliance-Herausforderungen in AI-gestützten Gesundheitssystemen, mit Fokus auf Differential Privacy und patientenzentrierte Datenverarbeitung; adressiert relevante technologische Ansätze wie Verschlüsselung und Differential Privacy sowie organisatorische und regulatorische Rahmenbedingungen—beinhaltet zudem die Herausforderungen bei der Integration von Blockchain-Technologien im healthcare-spezifischen Kontext und deren Vereinbarkeit mit der DSGVO, wodurch ein übergreifender Bezug zur Domäne Blockchain gegeben ist; SSI und PQC werden nicht explizit behandelt, ebenso steht die Anbindung an KRITIS außerhalb des engeren Fokus; bietet wertvolle Erkenntnisse zu datenschutzgerechter Systemgestaltung und Interdisziplinarität im Gesundheitswesen, bleibt aber hinsichtlich methodischer Tiefe und anwendungsbezogener Integration in allen vier Kernbereichen der Masterarbeit überblicksartig und konzeptionell.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
49	Tukur, M., Schneider, J., Househ, M., Dokoro, A. H., Ismail, U. I., Dawaki, M., & Agus, M. (2023). The metaverse digital environments: a scoping review of the challenges, privacy and security issues. <i>Frontiers in Big Data</i> , 1–25. https://doi.org/10.3389/fdata.2023.1301812	Niedrig	Umfassende Übersicht zu Herausforderungen, Datenschutz- und Sicherheitsfragen bei der Entwicklung und Implementierung von Metaverse-Umgebungen, insbesondere infolge der pandemiebedingten Digitalisierungsschübe; adressiert wirtschaftliche, technische, ethische und soziale Herausforderungen, darunter Hard- und Softwarekosten, digitale Ungleichheit sowie Regelwerks- und Datenmanagement-Fragen; konkrete Analyse und Klassifikation von Policy-, Privacy- und Security-Problemen, mit Fokus auf privatsphärenbezogene Risiken und Governance-Anforderungen im Metaverse. Explizite Bezüge zu SSI, Blockchain-Technologien und PQC fehlen; KRITIS werden nur implizit durch den Verweis auf digitale Spaltungen und gesellschaftliche Implikationen berührt.
50	Hanafi, B., Ali, M., & Singh, D. (2025). Quantum algorithms for enhanced educational technologies. <i>Discover Education</i> , 4(1), 1–33. https://doi.org/10.1007/s44217-025-00400-1	Niedrig	Fokus auf die Potenziale und Herausforderungen von Quantum Computing und Quantenkryptografie in der Bildungsbranche, z. B. für personalisiertes Lernen und sichere Datenübertragung; Bezug zu PQC nur anwendungsbezogen im Bildungskontext, ohne technische Tiefe oder Bezug zu SSI, Blockchain oder KRITIS; aus Sicht der vier Masterarbeitsdomänen methodisch und thematisch nur sehr eingeschränkt anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
51	Pillai, S. E. V. S., Nadella, G. S., Meduri, K., Priyadharsini, N. A., Bhuvanesh, A., & Kumar, D. (2025). A walrus optimization-enhanced long short-term memory model for credit fraud detection in banking. International Journal of Information Technology: An Official Journal of Bharati Vidyapeeth's Institute of Computer Applications and Management, 1–17. https://doi.org/10.1007/s41870-025-02574-1	Niedrig	Beschreibung einer innovativen Framework-Kombination aus Autoencoder, Long Short-Term Memory Netzwerken und Walrus Optimization Algorithm zur Verbesserung der Betrugserkennung im Bankensektor; konzentriert sich ausschließlich auf Machine-Learning-gestützte Analyse, Datenvorverarbeitung und Hyperparameteroptimierung zur Echtzeit-Erkennung betrügerischer Transaktionen in großen Datenmengen; keinerlei Behandlung oder Integration der vier Domänen der Masterarbeit; relevante methodische Beiträge beschränken sich auf KI-basierte Fraud Detection, ohne Anschlusspunkte zu den Kernthemen der Masterarbeit.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
52	Priya, S. S., Vijayabhasker, R., & Rajaram, A. (2025). Advanced Security and Efficiency Framework for Mobile Ad-Hoc Networks Using Adaptive Clustering and Optimization Techniques. <i>Journal of Electrical Engineering & Technology</i> (19750102), 20(3), 1815–1826. https://doi.org/10.1007/s42835-024-02119-9	Niedrig	Fokus auf ein innovatives Sicherheits- und Effizienz-Framework für Mobile Ad-Hoc Networks (MANETs) durch adaptive Clusterbildung, AI-unterstützte Vertrauensbewertung und quantenresistente PUF-Authentifizierung; explizite Relevanz für PQC durch QR-PUF-Komponente; SSI und Blockchain werden nicht behandelt, ebenso fehlt eine gezielte Betrachtung kritischer Infrastrukturen; für die vier Kerndomänen der Masterarbeit somit vor allem im Kontext quantensicherer Validierung/mobiler Netzwerksicherheit anschlussfähig, ansonsten methodisch und domänen spezifisch eingeschränkt.
53	Berkani, A.-S., Moumen, H., Benharzallah, S., Yahiaoui, S., & Bounceur, A. (2024). Blockchain Use Cases in the Sports Industry: A Systematic Review. <i>International Journal of Networked & Distributed Computing</i> , 12(1), 17–40. https://doi.org/10.1007/s44227-024-00022-3	Niedrig	Fokus auf branchenspezifische Anwendungen der Blockchain-Technologie im Sportsektor (Athleten-Datenmanagement, Fandaten, NFT-Sammlerstücke); methodische und technologische Vertiefung im Hinblick auf SSI, PQC oder KRITIS fehlt; relevante Erkenntnisse nur für den Bereich Blockchain-Anwendungsfälle in Sport und Entertainment, für die vier Domänen der Masterarbeit jedoch insgesamt wenig anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
54	2023 PNS Annual Meeting - Copenhagen, 17-20 June 2023. (2023). <i>Journal of the Peripheral Nervous System: JPNS</i> , 28 Suppl 4, S3–S254. https://doi.org/10.1111/jns.12585	Niedrig	Konferenzband ohne Bezug zu SSI, Blockchain oder PQC.
55	Posters. (2017). FEBS Journal, 284, 102–403. https://doi.org/10.1111/febs.14174	Niedrig	Posterband, kein Bezug zu SSI, Blockchain oder PQC.
56	Annotated Listing of New Books. (2024). <i>Journal of Economic Literature</i> , 62(4), 1696–1750. https://doi.org/10.1257/jel.62.4.1696	Niedrig	Buchliste, kein Bezug zum Thema.
57	PNS Abstracts 2023. (2023). <i>Journal of the Peripheral Nervous System</i> , 28, S3–S254. https://doi.org/10.1111/jns.12585	Niedrig	Abstractband, kein Bezug zu SSI, Blockchain oder PQC.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
58	Elendu, C., Omeludike, E. K., Oloyede, P. O., Obidigbo, B. T., & Omeludike, J. C. (2024). Legal implications for clinicians in cybersecurity incidents: A review. <i>Medicine</i> , 103(39), 1–26. https://doi.org/10.1097/MD.00000000000039887	Niedrig	Fokus auf die rechtlichen Implikationen von Cybersecurity-Vorfällen im Gesundheitswesen, insbesondere für klinisch tätige Personen; Betrachtung technologischer Entwicklungen (u. a. künstliche Intelligenz und Quantencomputing) sowie internationaler regulatorischer Unterschiede; praxisnahe Empfehlungen und Fallstudien zu Cybersecurity-Management, ethischen und juristischen Aspekten im Gesundheitssektor; keine explizite Behandlung von SSI, Blockchain oder PQC, KRITIS werden durch den Gesundheitsbereich berührt, Schwerpunkt liegt jedoch auf juristischen und ethischen Fragestellungen.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
59	Albshaier, L., Almarri, S., & Hafizur Rahman, M. M. (2024). A Review of Blockchain's Role in E-Commerce Transactions: Open Challenges, and Future Research Directions. Computers (2073-431X), 13(1), 27. https://doi.org/10.3390/computers13010027	Niedrig	Fokus auf die Anwendung von Blockchain-Technologien zur Verbesserung von Sicherheit, Transparenz und Betrugserkennung in E-Commerce-Transaktionen; betont die Rolle verteilter, unveränderlicher digitaler Ledger für den Schutz sensibler Kundendaten und die Stärkung des Vertrauens in Online-Plattformen; adressiert die Domäne „Blockchain“ vorrangig, ohne explizite Bezüge zu SSI oder PQC; KRITIS werden nicht thematisiert, da der Schwerpunkt auf E-Commerce liegt; methodische und technologische Tiefe für die vier Domänen der Masterarbeit ist auf Blockchain-Anwendungen im Bereich E-Commerce beschränkt.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
60	Reddy, R. C., Bhattacharjee, B., Mishra, D., & Mandal, A. (2022). A systematic literature review towards a conceptual framework for enablers and barriers of an enterprise data science strategy. <i>Information Systems & e-Business Management</i> , 20(1), 223–255. https://doi.org/10.1007/s10257-022-00550-x	Niedrig	Fokus liegt auf der systematischen Analyse von Erfolgsfaktoren und Hindernissen bei der unternehmensweiten Einführung von Data-Science-Strategien; methodische Entwicklung eines Enabler-Barrier-Frameworks für die erfolgreiche Umsetzung datengetriebener Projekte in Unternehmen; adressiert dabei organisatorische, technologische und strategische Aspekte der digitalen Transformation im breiten Kontext, jedoch ohne explizite Behandlung oder Integration von SSI, Blockchain-Technologien, PQC oder spezifischen Schutzanforderungen kritischer Infrastrukturen; liefert wertvolle Erkenntnisse zur Implementierung von Data Science im Unternehmensumfeld, ist für die vier Domänen der Masterarbeit methodisch und thematisch jedoch nicht anschlussfähig.

Fortsetzung auf nächster Seite

Tabelle A-8 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
61	Kumar, Y., Marchena, J., Awlla, A. H., Li, J. J., & Abdalla, H. B. (2024). The AI-Powered Evolution of Big Data. <i>Applied Sciences</i> (2076-3417), 14(22), 10176. https://doi.org/10.3390/app142210176	Niedrig	Fokus auf die Weiterentwicklung von Big-Data-Analyse und Management durch künstliche Intelligenz, mit Betonung neuer Rahmenwerke für die Charakterisierung und Handhabung großer, komplexer Datensätze. Der Beitrag stellt innovative AI-gestützte Tools (wie RAG-basierte Analyse-Bots/ChatGPT) zur Verbesserung der Datenanalyse vor und hebt methodologische Fortschritte im Bereich datengetriebene Entscheidungsunterstützung hervor. Keine explizite Behandlung oder Integration von SSI, Blockchain-Technologien, PQC oder besonderen Anforderungen kritischer Infrastrukturen; methodische und technologische Beiträge beschränken sich auf Big-Data-Management und AI-basierte Analytics, ohne Verknüpfung zu den vier zentralen Domänen der Masterarbeit.

Anmerkung. Basierend auf den Abstracts aller in Spalte zwei unter „Quelle“ aufgeführten Quellenangaben.

Anhang 2.2: Zweite Iteration vom 02. November 2025

Die zweite Iteration erfolgt im Rahmen der Masterarbeit unter Berücksichtigung der in der ersten Iteration definierten ausgewählten Methoden der PRISMA 2020 Richtlinien (Tabelle A-2).

Angepasst wurde der Suchzeitraum (Zeitrahmen: 30. Mai 2025 bis 02. November 2025) innerhalb der Ein- und Ausschlusskriterien, um die Wissensbasis zu aktualisieren und neu erschienene Publikationen im dynamischen Forschungsfeld der PQC und blockchain-basierten SSI-Systeme zu erfassen (Tabelle A-9).

Tabelle A-9

Ein- und Ausschlusskriterien für die systematische Literaturrecherche

Kategorie	Einschluss	Ausschluss	Begründung
Thematischer Fokus	SSI und dezentrale Identitätslösungen; Blockchain-basierte Identitätsmanagement-systeme; PQC und quantensichere Algorithmen; Sicherheit und Compliance in KRITIS; DSR-Methodik in IT/Informationssystemen; Kontext; Kryptoagilität und kryptografische Migration	Identitätsmanagement ohne Bezug zu SSI oder Blockchain; Klassische PKI ohne PQC-Bezug; Kryptografie ohne Post-Quantum-Relevanz; Arbeiten ohne Bezug zu KRITIS oder ohne sicherheitskritischen IT/Informationssystemen; Nicht-DSR-basierte Entwicklungsansätze	Fokussierung auf die Forschungsfragen und relevante technologische, methodische und regulatorische Aspekte.
Zeitrahmen	30. Mai 2025 bis 02. November 2025	Alles davor oder danach	Berücksichtigung aktueller technologischer Entwicklungen (Blockchain, PQC, SSI) und regulatorischer Anforderungen.

Fortsetzung auf nächster Seite

Tabelle A-9 (Fortsetzung)

Kategorie	Einschluss	Ausschluss	Begründung
Publikationstypen	Peer-reviewed Journalartikel; Konferenzbeiträge anerkannter Fachgesellschaften (z. B. IEEE, ACM, IFIP); Preprints; Offizielle Standards und Empfehlungen (z. B. NIST, W3C, BSI); Whitepaper etablierter Organisationen; Dissertationen und anerkannte Fachbücher	Blogposts, Forenbeiträge, Marketingmaterial; Po- pularwissenschaftliche Artikel ohne wissenschaftliche Fundierung; Unveröffentlichte Manuskripte ohne Peer-Review; Seminar- und Abschlussarbeiten ohne wissenschaftliche Begutachtung	Sicherstellung wissenschaftli- cher Qualität, Nachvollziehbar- keit und Relevanz der Quellen für die Masterarbeit; Da das For- schungsthema aktuell noch sehr neu ist und die einschlägige Fachliteratur teilweise noch nicht den Peer- Review-Prozess durchlaufen hat, werden auch Preprints in die Analyse einbezogen. Preprints ermöglichen eine zeitnahe Verfügbarkeit aktueller Forschungser- gebnisse, was insbesondere bei diesem innovativen von großer Bedeutung ist.

Fortsetzung auf nächster Seite

Tabelle A-9 (Fortsetzung)

Kategorie	Einschluss	Ausschluss	Begründung
Sprache	Deutsch; Englisch	Andere Sprachen als Deutsch und Englisch	Gewährleistung der Verständlichkeit und Zugänglichkeit für den deutsch- und englischsprachigen Forschungskontext.
Zugänglichkeit	Verfügbare Volltexte	Nicht verfügbare Volltexte	Ermöglichung einer gründlichen Analyse und Bewertung der Inhalte.

Anmerkung. Eigene Darstellung.

Um die Konsistenz und Vergleichbarkeit der Ergebnisse zu gewährleisten wurde die zweite Iteration mit identischer Suchstrategie durchgeführt. Dazu zählen alle in Tabelle A-4 dargestellten Schritte, von der Identifikation relevanter Schlüsselkonzepte bis zur Übersetzung der Suchanfrage für die EBSCO Datenbank (Tabelle A-5, Tabelle A-6, Listing A-1).

Übersetzung der Suchanfrage für EBSCO

Das Ergebnis der EBSCO Suchanfrage ist in Abbildung A-3 dargestellt. Insgesamt wurden mit dieser Abfrage 34 Quellen identifiziert.

Abbildung A-3
Zweite Iteration - Ergebnis der EBSCO Suchanfrage

The screenshot shows the EBSCO search results interface. At the top, there is a search bar with the query: ("self-sovereign identity" OR ssi) OR ("decentralized identity" OR "dezentrale Identität") OR "digitale Identität" OR "user-controlled identity" OR "verifiable wallet" OR "verifiable credentials". Below the search bar, there are filters for "Alle Filter (3)", "Verfügbare Volltexte", "Peer-Reviewed", and "30.05.2025 - 02.1...". The search results are displayed in a grid format.

Result 1: *Identity Management Systems: A Comprehensive Review.* (Peer-reviewed | Wissenschaftl. Zeitschrift)

Von: Feng, Zhengze; Li, Ziyi; Cui, Hui; +1 weitere • In: Information, Sep 2025, Bd. Jhdg. 16, ausgabe 9, Seite 778 • Complementary Index

Abstract: Blockchain technology has introduced new paradigms for **identity management** systems (IDMSs) enabling users to regain control over their **identity data** and reduce reliance on centralized authorities. In recent years, numerous **bio...** Mehr anzeigen

Themen: IDENTITY management systems; BLOCKCHAINS; APPLIED SCIENCES; RISK assessment; +2 weitere

Options: Zugriffsoptionen > [?] Mehr wie dieses

Result 2: *Smart Contracts, Blockchain, and Health Policies: Past, Present, and Future.* (Peer-reviewed | Wissenschaftl. Zeitschrift)

Von: Kurt, Kenan Kaan; Timurtaş, Meral; Pınar, Sevgan; +2 weitere • In: Information, Oct 2025, Bd. Jhdg. 16, ausgabe 10, Seite 853 • Complementary Index

Abstract: The integration of **blockchain technology** into healthcare systems has emerged as a technical solution for enhancing **data security**, protecting **privacy**, and improving interoperability. **Blockchain-based smart contracts** offer reliability... Mehr anzeigen

Themen: Administration of Public Health Programs; All Other Health and Personal Care Stores; Computer systems design and related services (except video game design and development); Other Computer Related Services; +7 weitere

Options: Zugriffsoptionen > [?] Mehr wie dieses

Anmerkung. Eigene Darstellung.

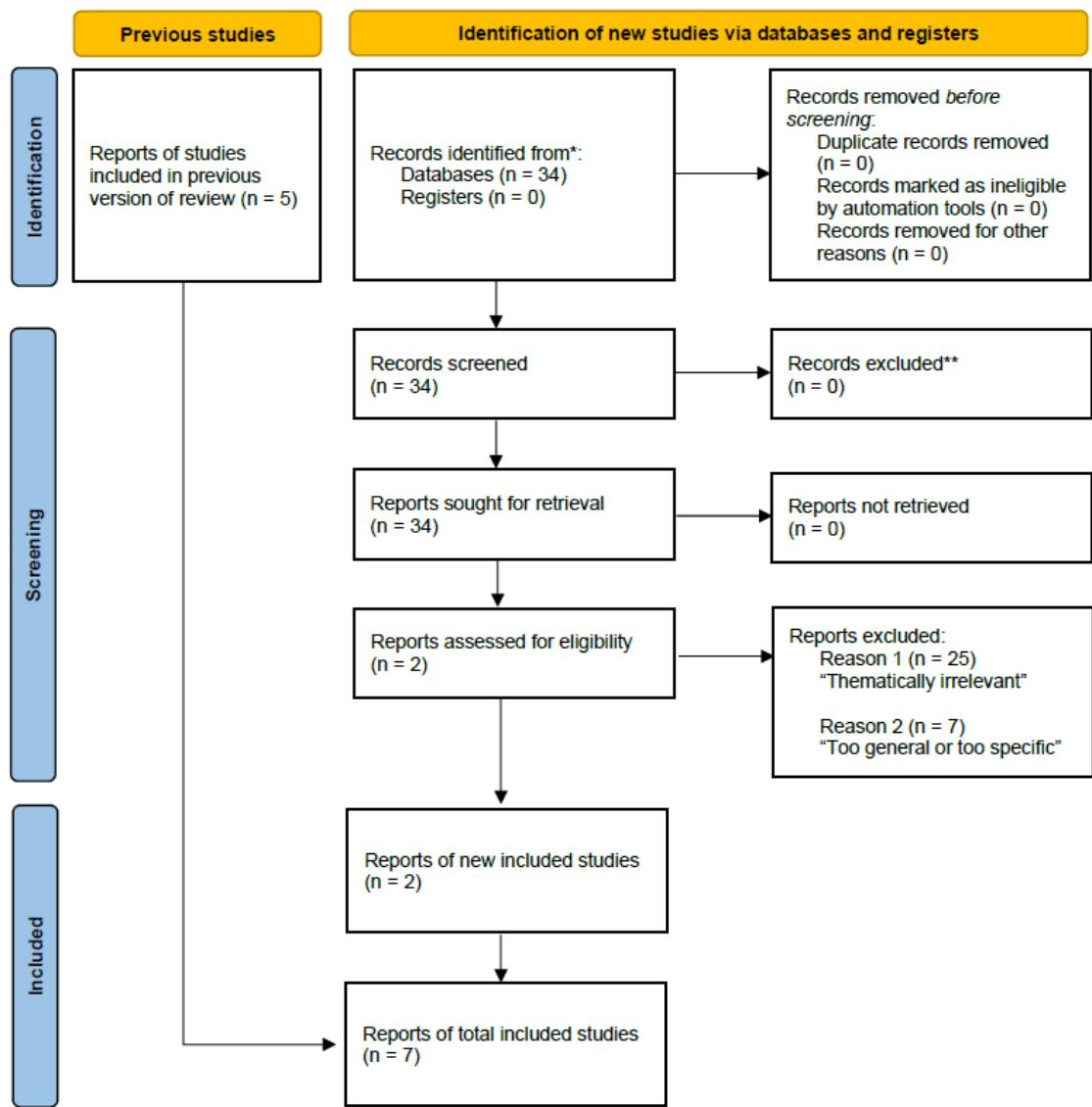
Selektionsprozess

Im nächsten Schritt folgt der Selektionsprozess der 34 identifizierten Quellen aufbauend auf den zuvor definierten Schlüsselkonzepten und der Suchstrategie. Dabei werden die in Tabelle A-9 definierten Ein- und Ausschlusskriterien konsequent angewendet, sodass nur fachlich einschlägige, qualitativ belastbare und für die Forschungsfragen zentrale Studien inkludiert werden.

Zunächst werden alle über die EBSCO-Suchanfrage identifizierten 34 Treffer einer Duplikatsbereinigung unterzogen und anschließend im Rahmen des Titel- und Abstract-Screenings hinsichtlich ihres thematischen Fokus bewertet. Hohe Relevanz erhalten Quellen mit klaren Beiträgen zu SSI, PQC, KRITIS oder dezentralen Identitätsarchitekturen, während Arbeiten zu angrenzenden Technologien wie Blockchain-Sicherheit im IoT oder digitaler Forensik als mittel relevant und allgemeine Technologietrends ohne direkten Bezug zum Thema als niedrig relevant eingestuft werden.

Abbildung A-4 fasst diesen Selektionsprozess der zweiten Iteration zusammen. Nach dem Screening aller 34 Publikationen folgte die Eignungsprüfung, in deren Ergebnis zwei Publikationen die Einschlusskriterien erfüllten und 32 ausgeschieden wurden, da sie entweder thematisch irrelevant ($n = 25$) oder hinsichtlich ihrer Spezifität unangemessen waren ($n = 7$). Insgesamt wurden somit im Rahmen beider Iterationen sieben relevante Quellen für die Masterarbeit identifiziert, die zur Synthese des aktuellen Standes der Forschung in Kapitel 1.2 eingeflossen sind.

Abbildung A-4
Zweite Iteration - PRISMA 2020 Flussdiagramm



Anmerkung. In Anlehnung an Page et al. (2021, S. 5).

Tabelle A-10 stellt eine Übersicht der Bewertung der 34 identifizierten Quellen dar, welche vollständig in Tabelle A-11 dokumentiert wurde.

Tabelle A-10

Zweite Iteration - Übersicht der relevanten Quellen

Nr.	Quelle	Relevanz
1	Barrett-danes, F., & Ahmad, F. (2025). Quantum computing and cybersecurity: a rigorous systematic review of emerging threats, post-quantum solutions, and research directions (2019-2024). <i>Discover Applied Sciences</i> , 7(10). https://doi.org/10.1007/s42452-025-07322-5	Hoch
2	Feng, Z., Li, Z., Cui, H., & Whitty, M. T. (2025). Identity management systems: A comprehensive review. <i>Information</i> (Basel), 16(9), 778. https://doi.org/10.3390/info16090778	Hoch
3–10	Diverse	Mittel
11–34	Diverse	Niedrig

Anmerkung. Basierend auf Tabelle A-11 und Titel und Abstracts von Barrett-danes und Ahmad (2025) und Feng et al. (2025).

Bewertung der identifizierten Quellen hinsichtlich ihrer Relevanz

Tabelle A-11

Zweite Iteration - Relevanzbewertung der identifizierten Quellen

Nr.	Quelle	Relevanz	Kommentar
1	Barrett-danes, F., & Ahmad, F. (2025). Quantum computing and cybersecurity: a rigorous systematic review of emerging threats, post-quantum solutions, and research directions (2019-2024). Discover Applied Sciences, 7(10). https://doi.org/10.1007/s42452-025-07322-5	Hoch	Setzt sich systematisch, methodisch und interdisziplinär mit den Bedrohungen durch Quantum Computing für klassische Kryptosysteme auseinander und behandelt ausdrücklich Post-Quantum-Kryptografie (PQC), hybride Frameworks (u.a. QKD), Umsetzungsherausforderungen und deren Auswirkungen insbesondere für IoT-Umgebungen. Die Arbeit analysiert sowohl den Forschungsstand als auch Implementierungs- und Migrationspfade anhand konkreter Pilotstudien und adressiert Skalierbarkeit und Wirtschaftlichkeit; Die Nutzung eines PRISMA-basierten Review-Frameworks und der Fokus auf praktische Handlungs- und Politikempfehlungen verleihen der Arbeit hohe wissenschaftliche und praxisnahe Relevanz für die Entwicklung, Migration und Absicherung quantensicherer sowie zukunftsfähiger Systeme mit starker Anschlussfähigkeit an KRITIS, PQC und angrenzende sicherheitskritische Domänen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
2	Feng, Z., Li, Z., Cui, H., & Whitty, M. T. (2025). Identity management systems: A comprehensive review. <i>Information</i> (Basel), 16(9), 778. https://doi.org/10.3390/info16090778	Hoch	Der Abstract beschreibt eine umfassende und systematische Übersicht zu Blockchain-basierten Identity Management Systems (IDMSs) mit explizitem Fokus auf Self-Sovereign Identity (SSI), dezentralen Identifikatoren (DIDs), Interoperabilität und Sicherheitsanalyse über den gesamten Identitätslebenszyklus hinweg. Die Arbeit adressiert technologische und organisatorische Herausforderungen, wie Revokation, Übertragbarkeit, Interoperabilität und Quantum-Resilienz für nutzerkontrollierte Identitätsmodelle. Die PRISMA-basierte Methodik sowie die sektorübergreifende Taxonomisierung und die systematische Ableitung aktuell ungelöster Probleme und künftiger Forschungsrichtungen machen die Quelle höchst wertvoll im Umfeld SSI, Blockchain, PQC und KRITIS.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
3	Akkal, M., Cherbal, S., Annane, B., Lakhlef, H., & Kharoubi, K. (2025). Quantum, post-quantum, and blockchain approaches for securing the internet of medical things: a systematic review. Cluster Computing, 28(10). https://doi.org/10.1007/s10586-025-05481-z	Mittel	Behandelt systematisch die Bedrohungen durch Quantum Computing für IoMT und fokussiert explizit auf den Schutz medizinischer Infrastrukturen mit quantensicherer Kryptografie und Blockchain-Technologie; analysiert sowohl PQC- als auch Blockchain-basierte Sicherheitslösungen und deren Anwendung in hochsensiblen medizinischen IoT-Umgebungen, adressiert Architektur, Herausforderungen und Lösungsansätze für KRITIS-nahe Gesundheitssektoren; Fokus zu stark auf IoMT ohne SSI- und ohne Implementierungsbezug.
4	Elkhodr, M. (2025). An AI-driven framework for integrated security and privacy in Internet of Things using quantum-resistant blockchain. Future Internet, 17(6), 246. https://doi.org/10.3390/fi17060246	Mittel	Der Abstract beschreibt ein ganzheitliches Framework, das KI-basierte Security-Orchestrierung, Blockchain-gestützte Identitätsverwaltung und quantenresistente Kryptografie explizit miteinander kombiniert und anwendungsnahe für IoT-Umgebungen evaluiert. Die Arbeit bietet indirekten Bezug zur technischen Umsetzung (Evaluierung) von PQC, SSI und KRITIS.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
5	Kameni Tcheumaga, F. N., Umba, K., Velupillaiameikandan, P., Haque, M. M., & Ahamed, S. I. (2025). Security of quantum federated machine learning with blockchain for electronics health records. <i>Cureus Journal of Computer Science.</i> https://doi.org/10.7759/s44389-025-03870-4	Mittel	Der Abstract adressiert systematisch die Integration von Quantum Federated Machine Learning (QFML), Blockchain und den Schutz elektronischer Gesundheitsakten, mit explizitem Fokus auf Security und Privacy in hochsensiblen, dezentralen Architekturen. Die Kombination aus quantenresistenter Kryptografie, Blockchain-basierter Validierung, federiertem Lernen und gezieltem Schutz kritischer Infrastrukturen im Gesundheitswesen wird als innovativer Lösungsansatz diskutiert. Die Arbeit analysiert bestehende Lücken hinsichtlich Robustheit und Sicherheit und liefert damit auch Impulse für zukünftige Forschung und Entwicklung in den Feldern PQC, KRITIS und dezentraler Identitätsarchitekturen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
6	Kurt, K. K., Timurtaş, M., Pınar, S., Ozaydin, F., & Türkeli, S. (2025). Smart contracts, blockchain, and health policies: Past, present, and future. <i>Information</i> (Basel), 16(10), 853. https://doi.org/10.3390/info16100853	Mittel	Der Abstract liefert eine methodisch fundierte, systematische Übersicht zu Blockchain- und Smart-Contract-basierten Lösungen für die Verwaltung von Gesundheitsdaten und das Policy Management im Gesundheitswesen. Explizit adressiert werden Sicherheit, Datenschutz, Interoperabilität und die Rolle von Smart Contracts in der Durchsetzung und Automatisierung digitaler Gesundheitsrichtlinien. Es fehlt ein expliziter Fokus auf Post-Quantum Kryptografie in Verbindung mit Self-Sovereign Identity.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
7	Qatawneh, M. (2025). A framework for security risk assessment of blockchain-based applications. <i>Indonesian Journal of Electrical Engineering and Computer Science</i> , 39(2), 952. https://doi.org/10.11591/ijeecs.v39.i2.pp952-962	Mittel	Der Abstract beschreibt ein systematisch entwickeltes und praxisvalidiertes Framework zur Bewertung und Mitigation von Sicherheitsrisiken in Blockchain-Anwendungen über alle Schichten hinweg. Es adressiert explizit kritische Schwachstellen wie Smart-Contract-Exploits, Sybil-Attacken und Private-Key-COMPROMISES, integriert quantitative und qualitative Risikoanalyse und demonstriert konkrete Wirksamkeit in Form signifikanter Risikoreduktionen in einem Ethereum-Case-Study-Szenario. Das resultierende BCRAM bietet einen standardisierten, adaptierbaren Bewertungsansatz, der für sichere Systemarchitektur in KRITIS-, SSI- und PQC-orientierten Blockchain-Projekten nutzbar ist.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
8	Reddy, N. R., Suryadevara, S., Reddy, K. G. R., Umamaheswari, R., Guttula, R., & Kotoju, R. (2025). Quantum secured blockchain framework for enhancing post quantum data security. <i>Scientific Reports</i> , 15(1), 31048. https://doi.org/10.1038/s41598-025-16315-8	Mittel	Der Abstract stellt ein hochinnovatives, ganzheitliches Framework („QuantumShield-BC“) für die Absicherung von Blockchain-Architekturen gegen die Bedrohungen durch Quantencomputer vor: Es integriert explizit Post-Quantum-Kryptografie (z.B. Dilithium, Falcon), Quantum Key Distribution (QKD) und einen Quantum Byzantine Fault Tolerance (Q-BFT) Konsens-Mechanismus auf Basis von Quantum Random Number Generation (QRNG) für Leader- und Validator-Auswahl. Das System zeigt in experimenteller Validierung sehr geringe Latenz, hohe Durchsatzraten und vollständige Immunität gegenüber typischen Quantum-Angriffen (Shor, Grover), einschließlich effektiver Sybil-, Replay- und MITM-Abwehr. Die Architektur demonstriert praktisch den Weg für interoperable, skalierbare, hochsichere Blockchains ohne direkten Bezug zu KRITIS und SSI.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
9	Tawfik, A. M., Al-Ahwal, A., Eldien, A. S. T., & Zayed, H. H. (2025). Blockchain-based access control and privacy preservation in healthcare: a comprehensive survey. <i>Cluster Computing</i> , 28(8). https://doi.org/10.1007/s10586-025-05308-x	Mittel	Die Arbeit bietet eine systematische Übersicht über Blockchain-basierte Ansätze zur Zugriffssteuerung und Wahrung der Privatsphäre im Gesundheitswesen, mit besonderem Fokus auf permissioned und permissionless Frameworks, Smart Contracts, kryptographische Verfahren und Plattformen wie Hyperledger Fabric und Ethereum. Die Analyse der untersuchten Lösungen zeigt, wie feingranularer Zugriff, automatisierte Autorisierung und revisionssichere Auditierung durch Blockchain und Privacy-Preserving-Techniken realisiert werden können. Es fehlen Bezüge zu SSI.
10	Aboshosha, B. W., Zayed, M. M., Khalifa, H. S., & Ramadan, R. A. (2025). Enhancing Internet of Things security in healthcare using a blockchain-driven lightweight hashing system. <i>Beni-Suef University Journal of Basic and Applied Sciences</i> , 14(1). https://doi.org/10.1186/s43088-025-00644-8	Niedrig	Fokussiert auf die Verbesserung der Datensicherheit und Integrität in IoT-basierten Gesundheitsanwendungen durch Blockchain und leichtgewichtige Hashverfahren; adressiert wesentliche Aspekte dezentraler Datenverwaltungsmodelle und kryptographischer Effizienz für ressourcenbeschränkte Geräte; PQC und SSI werden jedoch nicht behandelt, und der methodische Fokus liegt auf klassischen leichten Hashfunktionen statt quantensicheren Primitive.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
11	Addula, S. R. (2025). Mobile banking adoption: A multi-factorial study on social influence, compatibility, digital self-efficacy, and perceived cost among generation Z consumers in the United States. <i>Journal of Theoretical and Applied Electronic Commerce Research</i> , 20(3), 192. https://doi.org/10.3390/jtaer20030192	Niedrig	Konzentriert sich auf nutzerpsychologische und soziotechnische Faktoren zur Akzeptanz mobiler Banking-Anwendungen im FinTech-Bereich mit Fokus auf Konsumentenverhalten; weder Kryptografie, Blockchain, noch dezentrale Identitätssysteme werden thematisiert; auch fehlen technische oder sicherheitsbezogene Bezüge zu PQC, SSI oder KRITIS.
12	Al Jasem, M. S., De Clark, T., & Shrestha, A. K. (2025). Toward decentralized intelligence: A systematic literature review of blockchain-enabled AI systems. <i>Information</i> (Basel), 16(9), 765. https://doi.org/10.3390/info16090765	Niedrig	Die Arbeit liefert einen systematischen Überblick zu Blockchain-gestützten dezentralen KI-Systemen und adressiert damit relevante Aspekte dezentraler Architekturen, insbesondere durch die Untersuchung von Governance, Integrität, skalierbaren Konsensmechanismen sowie Sicherheits- und Datenschutzherausforderungen. Explizite technologische Bezüge zu Self-Sovereign Identity, Post-Quantum-Kryptografie oder kritischen Infrastrukturen fehlen allerdings; stattdessen steht das Zusammenwirken von Blockchain, Smart Contracts und KI im Fokus.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
13	Almazroi, A. A., Alqarni, M. A., Al-Shareeda, M. A., Alkinani, M. H., Almazroey, A. A., & Gaber, T. (2025). A bilinear pairing-based anonymous authentication scheme for 5G-assisted vehicular fog computing. Arabian Journal for Science and Engineering, 50(15), 11757–11778. https://doi.org/10.1007/s13369-024-09617-y	Niedrig	Die Arbeit konzentriert sich auf Authentifizierungsverfahren für 5G-unterstützte Vehicular Fog Computing-Systeme unter Nutzung bilinearer Parings; Im Mittelpunkt stehen effiziente Überprüfungsmechanismen, die Anonymität, Authentizität und praktische Sicherheit im Fahrzeugnetz verbessern; dezentrale Identitätsarchitekturen, Blockchain, PQC oder spezielle Konzepte für Self-Sovereign Identity oder den Schutz kritischer Infrastrukturen werden jedoch nicht behandelt.
14	Alsadie, D. (2025). Cybersecurity and artificial intelligence in unmanned aerial vehicles: Emerging challenges and advanced countermeasures. IET Information Security, 2025(1). https://doi.org/10.1049/ise2/2046868	Niedrig	Stellt eine fundierte und breit gefächerte Übersicht zu aktuellen Bedrohungen und fortschrittlichen Gegenmaßnahmen im Bereich AI-gestützter UAV-Systeme bereit und greift dabei gleich mehrere Schlüsseltechnologien des Bewertungsschemas direkt und explizit auf: Es werden konkrete Methoden der Post-Quantum-Kryptografie (PQC) und blockchain-basierte Sicherheitsmechanismen analysiert und hinsichtlich ihrer Wirksamkeit im Kontext hochsensibler und autonom agierender Dronennetze bewertet; Kein Bezug zu KRITIS & SSI;

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
15	Cavus, M., Ayan, H., Bell, M., & Dissanayake, D. (2025). Advances in energy storage, AI optimisation, and cybersecurity for electric vehicle grid integration. <i>Energies</i> , 18(17), 4599. https://doi.org/10.3390/en18174599	Niedrig	Die Arbeit liefert einen integrativen und interdisziplinären Überblick zu drei zentralen Säulen hochrelevanter Technologien: Fortschritte bei sicheren dezentralen Energiespeichern, KI-basierter Optimierung für Echtzeit-Energiemanagement und den Einsatz post-quantum Kryptografie und Blockchain-Systemen für die Absicherung von Vehicle-to-Grid (V2G)-Transaktionen im Smart-Grid-Kontext. Konkret fehlt es an praktischen Beiträgen bzgl. Implementierung von SSI, PQC und Blockchain für KRITIS.
16	A, C., & Basarkod, P. I. (2024). A survey on blockchain security for electronic health record. <i>Multimedia Tools and Applications</i> . https://doi.org/10.1007/s11042-024-19883-5	Niedrig	Der Abstract beschreibt eine systematische Übersicht zu Blockchain-basierten Sicherheitsansätzen für elektronische Gesundheitsakten (EHR), mit Schwerpunkt auf Datenschutz, Zugriffskontrolle, Integritätsprüfung und Effizienz, insbesondere auch mit Blick auf Deep-Learning-Ansätze im Healthcare-Umfeld. Zwar adressiert die Arbeit relevante Aspekte dezentraler vernetzter Datenstrukturen und gibt einen fundierten Vergleich vorhandener Blockchain-basierten Technologien für EHR-Sicherheit, doch stehen SSI, PQC oder KRITIS-spezifische Anwendungen nicht explizit im Fokus.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
17	Guayasamín, A., Fuertes, W., Carrera, N., Tello-Oquendo, L., & Suango, V. (2025). Blockchain-empowered e-ticket distribution system for secure and efficient transactions, validation, and audits. Annals of Telecommunications - Annales Des Télécommunications. https://doi.org/10.1007/s12243-025-01125-w	Niedrig	Der Abstract beschreibt ein Blockchain-basiertes System zur sicheren und effizienten Verwaltung, Prüfung und Auditierung von E-Tickets im Unterhaltungsbereich, insbesondere für Raffles und Veranstaltungsmanagement. Der Fokus liegt auf betrugsresistenter Ticketvergabe, Transparenz, digitaler Signierung, Hash-Funktionen und Proof-of-Work-basiertem Mining, was für die Optimierung von Fälschungsschutz und Automatisierung in öffentlichen Systemen relevant ist. Es besteht kein direkter fachlicher Bezug zu den Kerndomänen PQC, SSI, oder KRITIS.
18	He, X., Xu, G., Han, X., Wang, Q., Zhao, L., Shen, C., ... Feng, D. (2025). Artificial intelligence security and privacy: a survey. Science China Information Sciences, 68(8). https://doi.org/10.1007/s11432-025-4388-5	Niedrig	Der Abstract bietet eine breite, systematische Übersicht über Sicherheits- und Datenschutzherausforderungen von KI-Systemen, inklusive Bedrohungen für Datenintegrität, Trainings-, Inferenzphasen und verteilter Umgebungen. Zentral sind klassische KI-Attacken wie Datenvergiftung, Backdoor und Adversarial Attacks, jedoch fehlt ein expliziter Bezug zu Self-Sovereign Identity, Blockchain-Integration, Post-Quantum Kryptografie oder KRITIS-spezifischen Sicherheitsarchitekturen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
19	<p>Khagga, V., Priya., S., & Prasad, A. M. (2025). Enhanced QoS-aware secure routing protocol for WAHNs using advanced fast double decker new binary archimedes kepler pure convolutional transformer network and cryptographic techniques. <i>Peer-to-Peer Networking and Applications</i>, 18(4). https://doi.org/10.1007/s12083-025-02035-3</p>	Niedrig	<p>Der Abstract stellt eine klassische netzwerktechnische Studie dar, die ein innovatives Routing-Protokoll für Wireless Ad-Hoc Networks (WAHNs) vorstellt und dabei verschiedene Optimierungsverfahren, Deep-Learning-Modelle und komplexe kryptografische Techniken integriert. Die Arbeit adressiert primär technische Fragen der Effizienzsteigerung, Clusterbildung, Verzögerungsminimierung und Zugangssicherheit auf Netzwerkebene, beschränkt sich aber methodisch auf klassische und KI-gestützte kryptografische Mechanismen ohne expliziten Bezug zu PQC, Blockchain, SSI oder KRITIS-spezifischen Identitätsbeziehungsweise Infrastrukturarchitekturen.</p>

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
20	Kumar, M., Kaur, G., & Rana, P. S. (2025). Performance, portability, productivity, and security in HPC cloud: a systematic literature review. <i>The Journal of Supercomputing</i> , 81(11). https://doi.org/10.1007/s11227-025-07685-x	Niedrig	Der Abstract liefert eine umfassende systematische Übersicht aktueller Entwicklungstrends und Herausforderungen im Bereich Cloud-basierter Hochleistungsrechner (HPC), strukturiert entlang der Aspekte Performance, Portabilität, Produktivität und Security. Während das Security-Kapitel innovative Technologien wie Trusted Execution Environments, Verschlüsselung und feingranulare Zugangskontrolle behandelt und so auf zentrale Anforderungen in sensiblen Anwendungsdomänen eingeht, fehlt ein expliziter Fokus auf Post-Quantum Kryptografie, Self-Sovereign Identity oder dezentrale Identitätsarchitekturen im KRITIS-Kontext.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
21	Lubis, M., Safitra, M. F., Niedrig Fakhrurroja, H., & Muttaqin, A. N. (2025). Guarding our vital systems: A metric for critical infrastructure cyber resilience. Sensors (Basel, Switzerland), 25(15), 4545. https://doi.org/10.3390/s25154545	Niedrig	Dieses Paper adressiert direkt die Entwicklung, Messung und Erhöhung der Cyber-Resilienz für kritische Infrastrukturen anhand des InfraGuard Cybersecurity Frameworks, welches etablierte Reifegradmodelle (ISO/IEC 15504, NIST CSF, COBIT) einbindet und die gesamte Breite an Abwehr-, Verteidigungs- und Wiederherstellungsmaßnahmen strukturiert abbildet. Im Mittelpunkt stehen situative Awareness, aktive Verteidigung, Risikomanagement und Incident Recovery. Es fehlt ein expliziter Fokus auf Post-Quantum Kryptografie in Verbindung mit Self-Sovereign Identity.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
22	Marengo, A., & Santamato, V. (2025). Quantum algorithms and complexity in healthcare applications: a systematic review with machine learning-optimized analysis. <i>Frontiers in Computer Science</i> , 7(1584114). https://doi.org/10.3389/fcomp.2025.1584114	Niedrig	Die Studie bietet eine systematische Übersicht zu quantencomputergestützten Algorithmen, deren algorithmischer Komplexität und Anwendungsbereichen im Gesundheitswesen, insbesondere für KI-gestützte Diagnostik und die Sicherheit medizinischer Daten. Sie kombiniert fortgeschrittene Literaturanalyse (PSO, LDA, LIME) mit einer empirisch validierten Kategorisierung der Forschung in „Quantum für KI in Healthcare“ und „Quantum für Datensicherheit im Gesundheitswesen“. Besonderes Gewicht liegt auf quantenkryptografischen Protokollen, Blockchain-basierten Sicherheitsarchitekturen und hybriden Quantum-KI-Ansätzen, die innovative Lösungsansätze für den Schutz sensibler Gesundheitsdaten, die Beschleunigung diagnostischer Prozesse und die Entwicklung resilenter Infrastrukturmodelle liefern. Es fehlt ein konkreter Bezug zu SSI und KRITIS.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
23	Meka, C., Palakollu, K. R., Azees, M., Rajasekaran, A. S., Das, A. K., & Hölbl, M. (2025). A comprehensive survey on integration of machine learning with secure blockchain-based applications. <i>Cluster Computing</i> , 28(10). https://doi.org/10.1007/s10586-025-05330-z	Niedrig	Der Abstract beschreibt eine systematische Übersicht zu Anwendungen, Herausforderungen und offenen Problemen an der Schnittstelle von Machine Learning und Blockchain, insbesondere hinsichtlich Sicherheit, Automatisierung, Auditing und Skalierung in IoT- und Industrieszenarien. Der Beitrag legt den Fokus auf die beidseitigen Synergien beider Technologien—z.B. Fraud Detection, Schutz der Modellintegrität, sichere Prozessautomatisierung per Smart Contracts und Anomalieerkennung—and skizziert aktuelle Forschung, technische Integrationsaspekte und sektorübergreifende Use Cases. Explizite Bezüge zu PQC, Self-Sovereign Identity oder KRITIS-spezifischen Identitätsbeziehungsweise Infrastrukturarchitekturen werden jedoch nicht vertieft adressiert.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
24	Mustafa, R., Sarkar, N. I., Mohaghegh, M., Pervez, S., & Vohra, O. (2025). Cross-layer analysis of machine learning models for secure and energy-efficient IoT networks. Sensors (Basel, Switzerland), 25(12), 3720. https://doi.org/10.3390/s25123720	Niedrig	Die Arbeit verfolgt einen innovativen Ansatz zur Harmonisierung von IoT-Sicherheit und Energieeffizienz durch eine umfassende, schichtübergreifende Architektur, die spezialisierte Machine-Learning-Modelle (z. B. LSTM-Netzwerke zur Anomalieerkennung, Entscheidungsbäume zur Validierung) mit leichtgewichtiger, adaptiver Kryptografie (Speck) koppelt. Durch rollenspezifische Zugriffskontrolle (RBAC) und energieorientierte Sicherheitspolitiken adressiert die Studie zentrale Herausforderungen für ressourcenbeschränkte IoT-Geräte und demonstriert signifikante Verbesserungen bei Fehlalarmraten, Zugangssicherheit und Energieverbrauch. Explizite Bezüge zu PQC, Self-Sovereign Identity oder KRITIS-spezifischen Identitätsbeziehungsweise Infrastrukturarchitekturen werden jedoch nicht vertieft adressiert.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
25	Mutahhar, A., Khanzada, T. J. S., & Shahid, M. F. (2025). Enhanced scalability and security in blockchain-based transportation systems for mass gatherings. Information (Basel), 16(8), 641. https://doi.org/10.3390/info16080641	Niedrig	Der Abstract beschreibt ein Blockchain-basiertes System zur Erhöhung von Effizienz und Sicherheit in intelligenten Transportsystemen, insbesondere für Massengroßveranstaltungen und städtische Mobilitätsnetzwerke. Die Lösung integriert State Channels und Rollups zur Skalierungsoptimierung, erreicht hohe Transaktionsgeschwindigkeiten und adressiert die Herausforderungen der Datenmanipulation, Integrität und Verschlüsselung in urbanen Verkehrsnetzen. Die Konzeption ist praxisnah für dynamische, datenschutzsensitive Transportinfrastrukturen, adressiert aber weder explizit Self-Sovereign Identity, Post-Quantum Kryptografie noch Anwendungsfälle für KRITIS-nahe Domänen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
26	Pillai, S. E. V. S., Nadella, G. S., Meduri, K., Priyadharsini, N. A., Bhuvanesh, A., & Kumar, D. (2025). A walrus optimization-enhanced long short-term memory model for credit fraud detection in banking. International Journal of Information Technology. https://doi.org/10.1007/s41870-025-02574-1	Niedrig	Der Abstract beschreibt ein fortgeschrittenes Framework zur Nutzung von Autoencoder, LSTM-Netzwerken und Walrus Optimization Algorithm (WOA) für die Optimierung von Kreditkartenbetrugserkennung in Bankdaten. Die Kombination von modernen Machine-Learning-Techniken und bio-inspirierten Metaheuristiken erhöht die Performance und Skalierbarkeit im Datenmanagement klassischer Finanzsysteme, bleibt aber ohne expliziten Bezug zu Blockchain, Self-Sovereign Identity, Post-Quantum Kryptografie, kritischer Infrastruktur oder dezentralen Identitätsarchitekturen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
27	Rao, C. K., Sahoo, S. K., & Yanine, F. F. (2025). A review of IoT-based smart energy solutions for photovoltaic systems. <i>Electrical Engineering</i> (Berlin. Print), 107(12), 15049–15068. https://doi.org/10.1007/s00202-025-03312-3	Niedrig	Der Abstract gibt einen breiten Überblick über den Stand und die Rolle von IoT-basierten Monitoring- und Managementsystemen für Photovoltaik-Anlagen, insbesondere zur Optimierung von Energieeffizienz, Datenanalyse, Cloud-Integration und Betriebsplanung. Der Schwerpunkt liegt auf technologischen Innovationen rund um IoT, Smart Grids, Energieverwaltung und Echtzeitdatenerfassung für industrielle und wissenschaftliche Anwendungen. Methodisch und thematisch fehlt jedoch ein Bezug zu Self-Sovereign Identity, Blockchain, Post-Quantum Kryptografie oder zur Absicherung kritischer digitaler Versorgungsinfrastrukturen.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
28	Sefati, S. S., Arasteh, B., Halunga, S., & Fratu, O. (2025). A comprehensive survey of cybersecurity techniques based on quality of service (QoS) on the Internet of Things (IoT). Cluster Computing, 28(12). https://doi.org/10.1007/s10586-025-05449-z	Niedrig	<p>Der Abstract liefert einen systematischen Überblick zu aktuellen Cybersecurity-Techniken zur QoS-bewussten Absicherung von IoT-Systemen, mit besonderem Fokus auf die Trade-offs zwischen Sicherheitsmechanismen (etwa Anomalieerkennung, Hybrid- und KI-basierte Methoden, Blockchain, Federated Learning) und kritischen Leistungsparametern (Latenz, Durchsatz, Energieverbrauch) in ressourcenbeschränkten Umgebungen.</p> <p>Der Beitrag beleuchtet vielseitige Angriffsszenarien (DDoS, MITM, Datenextraktion), validiert aktuelle Detektionsparadigmen und hebt offene Forschungsfelder hervor, etwa adaptive und echtzeitfähige Security-Modelle sowie Recovery-Maßnahmen nach Angriffen. Explizite Bezüge zu PQC, SSI oder KRITIS-spezifischen Architekturparadigmen werden nicht gesetzt.</p>

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
29	<p>Shahzad, M., Rizvi, S., Khan, T. A., Ahmad, S., & Ateya, A. A. (2025). An exhaustive parametric analysis for securing SDN through traditional, AI/ML, and blockchain approaches: A systematic review. <i>International Journal of Networked and Distributed Computing</i>, 13(1).</p> <p>https://doi.org/10.1007/s44227-024-00055-8</p>	Niedrig	<p>Das Paper bietet eine umfassende, systematische Analyse und Vergleich der wichtigsten Sicherheitsansätze für Software-defined Networking (SDN), inklusive klassischer, Machine-Learning- und Blockchain-basierter Methoden. Die Literaturauswertung demonstriert, dass Blockchain-basierte Mechanismen für Flussregelung, Datenvalidierung, Parser und Controller-Authentifizierung die Robustheit und Angriffsresistenz von SDN signifikant steigern und Aspekte wie Integrität, Trust und Dezentralität adressieren.</p> <p>Machine-Learning-Technologien (CNN, SVM, KNN) liefern praxisnahe Mehrwerte bei der Angriffserkennung. Jedoch bleibt ein expliziter Bezug zu Post-Quantum Kryptografie, Self-Sovereign Identity oder KRITIS-spezifischen Identitätsarchitekturen aus.</p>

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
30	Singh, B., Indu, S., & Majumdar, S. (2025). Comparative analysis of intrusion detection models using quantum machine learning techniques. Circuits, Systems, and Signal Processing. https://doi.org/10.1007/s00034-025-03256-w	Niedrig	Der Abstract und die zitierten Studien zeigen, dass Quantum Machine Learning-Techniken bei der Intrusion Detection gegenüber klassischen Machine Learning-Ansätzen, insbesondere in großen Netzwerken und komplexen Datenlagen, signifikante Vorteile in Genauigkeit und Performanz bieten können. Die experimentellen Vergleiche (z.B. QSVM, QCNN, VQC) über mehrere bekannte Datensätze belegen, dass quantenunterstützte Modelle wie das QML-IDS eine robustere Erkennung von Angriffsmustern (z.B. DDoS, BruteForce, Reconnaissance) und eine bessere Generalisierbarkeit im Kontext von Post-Quantum Cryptography und modernen Cyberbedrohungen erreichen. Auch werden praktische Herausforderungen und die Skalierbarkeit hybrider Modelle thematisiert. Ein expliziter Bezug zu Post-Quantum Kryptografie, Self-Sovereign Identity oder KRITIS-spezifischen Identitätsarchitekturen bleibt aus.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
31	Tom, A. K., Khraisat, A., Jan, T., Whaiduzzaman, M., Nguyen, T. D., & Alazab, A. (2025). Survey of federated learning for cyber threat intelligence in industrial IoT: Techniques, applications and deployment models. Future Internet, 17(9), 409. https://doi.org/10.3390/fi17090409	Niedrig	Die Arbeit liefert einen detaillierten und breit gefächerten Überblick zu aktuellen Methoden für cyber threat intelligence (CTI) im Kontext industrieller IoT-Umgebungen (IIoT) unter expliziter Berücksichtigung von föderiertem Lernen (FL) als Schlüsseltechnologie für datenschutzwahrende, skalierbare und dezentrale Bedrohungsanalyse. Die systematische Betrachtung von FL-Architekturen, Aggregationsstrategien (z.B. FedAvg, FedProx, Krum) und deren Anwendungen auf Intrusion Detection, Malware-Analyse, Botnet-Mitigation und Anomalieerkennung demonstriert praxisrelevante Fortschritte für KRITIS-nahe und industriell ausgerichtete Sicherheitsarchitekturen. Es fehlen Bezüge zu SSI, Blockchain und PQC.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
32	Zhang, Y., Zhao, K., Yang, Y., & Zhou, Z. (2025). Real-time service migration in edge networks: A survey. <i>Journal of Sensor and Actuator Networks</i> , 14(4), 79. https://doi.org/10.3390/jsan14040079	Niedrig	Der Abstract stellt eine umfassende systematische Übersicht zu Echtzeit-Service-Migration in Edge-Netzwerken bereit, einschließlich Architekturen, Modellen, Motivationen, Techniken und Anwendungsszenarien (z.B. Smart Cities, Smart Homes, Smart Manufacturing). Der Schwerpunkt liegt auf Algorithmen, Modellen und Methoden zur Reduktion von Latenz, Lastverteilung und dynamischer Ressourcenzuteilung für zeitkritische, verteilte Dienste. Sicherheit, Privacy, PQC, Blockchain oder dezentrale Identitätsarchitekturen werden nicht explizit adressiert. Der Beitrag ist für Netzwerkarchitektur und Echtzeitfähigkeit relevant, liefert aber keinen direkten methodischen oder konzeptionellen Beitrag zu SSI/PQC/KRITIS.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
33	Zinabu, N. G., Marye, Y. W., Tune, K. K., & Demilew, S. A. (2025). Comprehensive analysis of lightweight cryptographic algorithms for battery-limited Internet of Things devices. International Journal of Distributed Sensor Networks, 2025(1). https://doi.org/10.1155/dsn/9639728	Niedrig	Die Arbeit liefert eine systematische und breit angelegte Analyse zu aktuellen Lightweight-Kryptografie-Algorithmen speziell für energie- und ressourcenbeschränkte IoT-Geräte. Verglichen werden dabei unter anderem Algorithmen wie ASCON, SPECK, PRINCE, TWINE und modifizierte Varianten von AES-128 hinsichtlich Effizienz, Sicherheit, Energieverbrauch und Implementierungskomplexität in konkreten Hardware- und Softwareszenarien. Besonders hervorgehoben werden die trade-offs zwischen Durchsatz, Speicher- und Energiebedarf sowie Angriffstoleranz. Der systematische Review zeigt, dass etwa ASCON als sichere, performante Allround-Lösung für Authenticated Encryption anerkannt ist, während SPECK in Szenarien mit höchsten Durchsatzanforderungen durch Einfachheit punktet, jedoch unter größerer kritischer Scrutiny in puncto langfristiger Sicherheit steht. PQC- oder SSI-Verfahren werden in diesem Kontext nicht umfassend adressiert.

Fortsetzung auf nächster Seite

Tabelle A-11 (Fortsetzung)

Nr.	Quelle	Relevanz	Kommentar
34	Zreikat, A. I., AlArnaout, Z., Abadleh, A., Elbasi, E., & Mostafa, N. (2025). The integration of the Internet of Things (IoT) applications into 5G networks: A review and analysis. Computers, 14(7), 250. https://doi.org/10.3390/computers14070250	Niedrig	Die Arbeit bietet einen fundierten Review und eine Analyse zur Integration von IoT-Anwendungen in 5G-Netzwerke mit Schwerpunkt auf Konnektivität, Datenrate, Latenz, Interoperabilität und Anwendungsvielfalt (Smart Cities, Industrie 4.0, Healthcare etc.). Methodisch werden technische Potenziale (wie Network Slicing, Edge Computing, massive Machine-Type-Kommunikation) und Herausforderungen (Sicherheit, Energie, Netzmanagement) systematisch aufgearbeitet. Zwar werden Security und Privacy angesprochen, jedoch gibt es keinerlei dezidierte Analyse zu PQC, Self-Sovereign Identity, Blockchain, dezentrale Identitätsarchitekturen oder KRITIS-spezifische Schutzmaßnahmen.

Anmerkung. Basierend auf den Abstracts aller in Spalte zwei unter „Quelle“ aufgeführten Quellenangaben.

Anhang 3: Erste Iteration der Artefaktentwicklung

Anhang 3.1: Framework- und Technologie-Auswahl

Anhang 3.1.1: DLT-Plattform

Die Auswahl der DLT-Plattform für die Implementierung des PQC-SSI-Prototypen erfolgte unter Berücksichtigung der in Kapitel 4.2 definierten funktionalen Anforderungen sowie der KRITIS-Compliance-Anforderungen an SSI-Systeme. Wie von Ghosh et al. (2021, S. 3) demonstriert, setzen dezentrale Identitätsverwaltungssysteme auf existierende Frameworks als technologische Bausteine.

Für die vorliegende Arbeit wurde Hyperledger Indy als DLT-Plattform gewählt, da es als einzige Enterprise-Blockchain-Lösung explizit für Self-Sovereign Identity konzipiert wurde (Su & Hsu, 2023, S. 547–548).

Hyperledger Indy bietet eine permissioned Ledger-Architektur (Dixit, Smith-Creasey & Rajarajan, 2022, Tabelle 1), die offene Leseoperationen ermöglicht, während Schreibrechte auf designierte Trust Anchors (Ghosh et al., 2021, S. 4) beschränkt bleiben, die als autorisierte Akteure DID-Registrierungen und Credential-Schemata publizieren können.

Die Plattform implementiert nativ das VC-Model mit den vier zentralen Attributen Issuer-Identität, Holder-Kontrolle, manipulationssichere Claims und Revocation-Status (Su & Hsu, 2023, S. 548). Hyperledger Indy unterstützt kryptographische Akkumulatoren als Revocation Registries, die Privacy-Preserving Non-Revocation Proofs via Zero-Knowledge-Verfahren ermöglichen (Su & Hsu, 2023, S. 548).

Für die konkrete Implementierung wurde das von-network (bcgov, n. d. b) als containerisierte Indy-Deployment-Lösung eingesetzt. Das von-network stellt eine vorkonfigurierte Indy-Node-Pool-Infrastruktur bereit, bestehend aus vier Validator-Nodes, einem Genesis-Webserver zur Bereitstellung der Pool-Transaktionsdatei sowie einem Ledger-Browser für Transparenz- und Audit-Zwecke (bcgov, n. d. b).

Anhang 3.1.2: SSI-Framework

Für die Implementierung der SSI-Agenten wurde Hyperledger ACA-Py (openwallet-foundation, n. d. n) als Framework ausgewählt. ACA-Py (openwallet-foundation, n. d. n) ist eine produktionsreife, quelloffene SSI-Agenten-Infrastruktur, die es ermöglicht, VC-Ökosysteme als Issuer, Holder oder Verifier zu implementieren (Ghosh et al., 2021, S.

3, 7). Die Kombination von Hyperledger Aries und Hyperledger Indy hat sich als zentrale Toolchain für die Entwicklung von SSI-Anwendungen etabliert (Ferdous, Ionita & Prinz, 2022, S. 5).

Die Architektur von ACA-Py (openwallet-foundation, n. d. n) basiert auf einer Trennung zwischen Agent und Controller. Der Agent verwaltet alle Aries-Kernfunktionalitäten wie die Interaktion mit anderen Agents, sichere Speicherverwaltung und Event-Benachrichtigungen, während der Controller das spezifische Verhalten der Agenten-Instanz definiert (Bahce & Utku, 2023, S. 10). Der Agent stellt dem Controller eine REST-API für administrative Nachrichten bereit, und der Controller registriert einen Webhook beim Agent, um HTTP-Callbacks für Event-Benachrichtigungen zu empfangen (Bahce & Utku, 2023, S. 10–11).

Ein weiteres zentrales Feature ist die Unterstützung von DIDComm-Messaging-Protokollen, die asynchrone, Ende-zu-Ende-verschlüsselte Kommunikation zwischen Agents ermöglichen, wobei Nachrichten über intermediäre Agent-Konfigurationen geroutet werden (Bahce & Utku, 2023, S. 10).

Im Vergleich zu alternativen Frameworks wie Aries Go wurde ACA-Py ausgewählt, da Aries Go noch nicht produktionsbereit ist (Bahce & Utku, 2023, S. 13–14). Die Entscheidung für ACA-Py ermöglicht zudem die Nutzung des Aries-Askar-Wallets als Speicher-Backend, welches im Vergleich zum älteren Indy-Wallet signifikante Performance-Verbesserungen bietet (Bahce & Utku, 2023, S. 13).

Anhang 3.1.3: Kryptografiebibliothek

Die Auswahl von liboqs als zentrale Kryptografiebibliothek wird durch das Konzept des OQS Projekts motiviert, welches Stebila und Mosca (2017, S. 30–33) als Ansatz zur Prototypisierung quantenresistenter Kryptografie mit dem Ziel des zuverlässigen flächendeckenden Einsatzes entworfen hat. Stebila und Mosca (2017, S. 30) definiert OQS als open-source Softwareframework bestehend aus liboqs, einer C-Bibliothek mit quantenresistenten Algorithmen, und deren Integration in etablierte Protokolle wie OpenSSL.

Die Bibliothek bietet eine gemeinsame Schnittstelle für Schlüsselaustausch und digitale Signaturschemen (Stebila & Mosca, 2017, S. 31) sowie Implementierungen verschiedener Post-Quantum-Algorithmen (Solavagione & Vesco, 2025, S. 3–4, Tab. 1).

Sikeridis, Kampanakis und Devetsikiotis (2020, S. 5) demonstrieren die praktische Integration von liboqs-basierten Algorithmen in das OQS OpenSSL Fork für TLS 1.3. Ihre Ergebnisse zeigen, dass die liboqs-Abstraktionsschicht es ermöglicht verschiedene PQ-Signaturen in X.509-Zertifikaten zu integrieren (Sikeridis, Kampanakis & Devetsikiotis, 2020, S. 11–14).

Anhang 3.1.4: Revocation-Infrastruktur

Für die Revocation-Anforderung wird das Revocation-Schema von Hyperledger Indy (Anhang 3.1.1) gewählt. Indy implementiert Privacy-Preserving-Revocation (Fraser & Schneider, 2025, S. 1) durch kryptografische Akkumulatoren, wodurch Holder Nicht-Widerrufung via ZKP nachweisen können, ohne ihre Credential-IDs oder Indizes preiszugeben (Hardman, 2018).

Der Indy Tails Server (bcgov, n. d. a) stellt einen dedizierten File-Server für die Speicherung und Distribution von Revocation-Registry-Tails-Dateien bereit (bcgov, n. d. d). Er fungiert als zentraler Storage-Service, auf den sowohl Credential-Issuer (zum Upload der Tails-Dateien) als auch Verifier (zum Download für Proof-Verifikation) zugreifen (bcgov, n. d. d). Während die speicherintensiven Tails-Dateien physisch auf dem Tails-Server liegen, wird lediglich ihr kryptografischer Hashwert fileHash in der Revocation Registry Definition auf dem unveränderlichen Ledger persistiert (bcgov, n. d. d).

Die BC-Government-Referenzimplementierung bietet native Integration mit ACA-Py („ACA-Py Docs“, n. d.) und stellt sicher, dass Holder und Verifier die notwendigen kryptografischen Daten abrufen können.

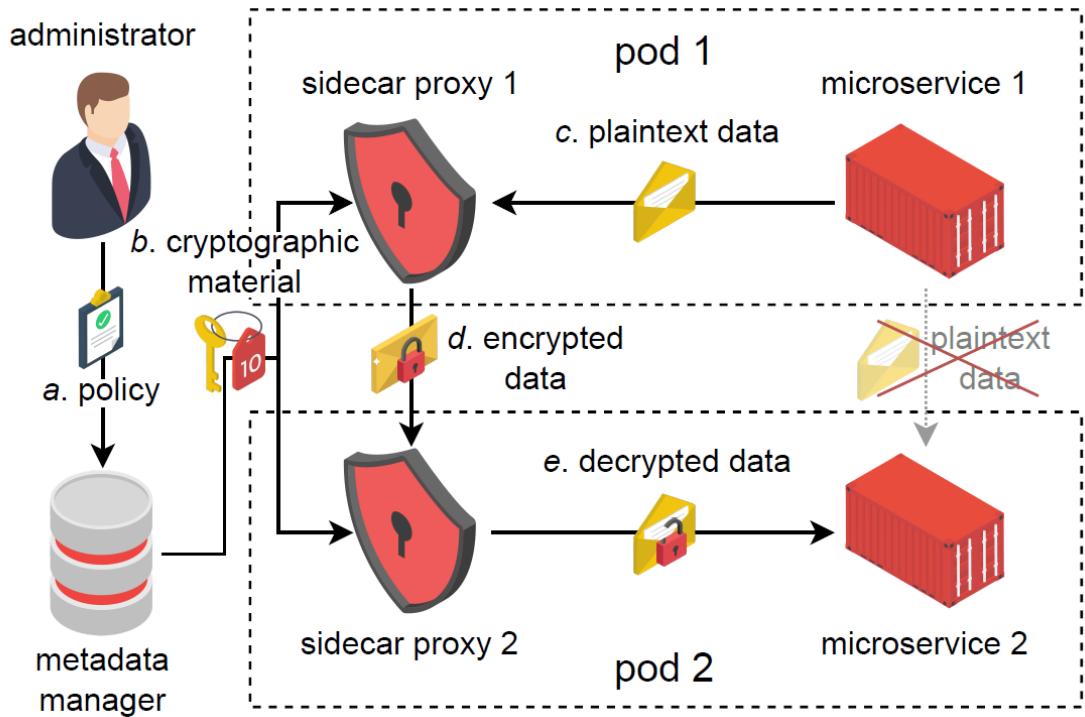
Anhang 3.1.5: Sidecar-Proxy

Das Sidecar-Proxy-Pattern wird zur transparenten Integration der PQC auf Transportebene gewählt, ohne die zugrundeliegende ACA-Py-Anwendungslogik zu modifizieren. Berlato et al. (2024, S. 2, 5) identifizieren Transparenz, Automatisierung und Modularität als zentrale Eigenschaften dieses Architekturmusters. Die vorliegende Arbeit adaptiert dieses Pattern von Cryptographic Access Control auf die PQC-Integration für SSI-Systeme.

Durch die Deployment-seitige Ergänzung von Sidecar-Containern können unterstützende Funktionalitäten für alle Services zentral implementiert werden, ohne die Komplexität einzelner Microservices zu erhöhen (Meadows et al., 2023, S. 158).

Abbildung A-5 visualisiert die grundlegende Funktionsweise eines Sidecar-Proxy innerhalb eines Pods.

Abbildung A-5
Pod mit Sidecar-Proxy



Anmerkung. Aus Berlato et al. (2024, S. 3).

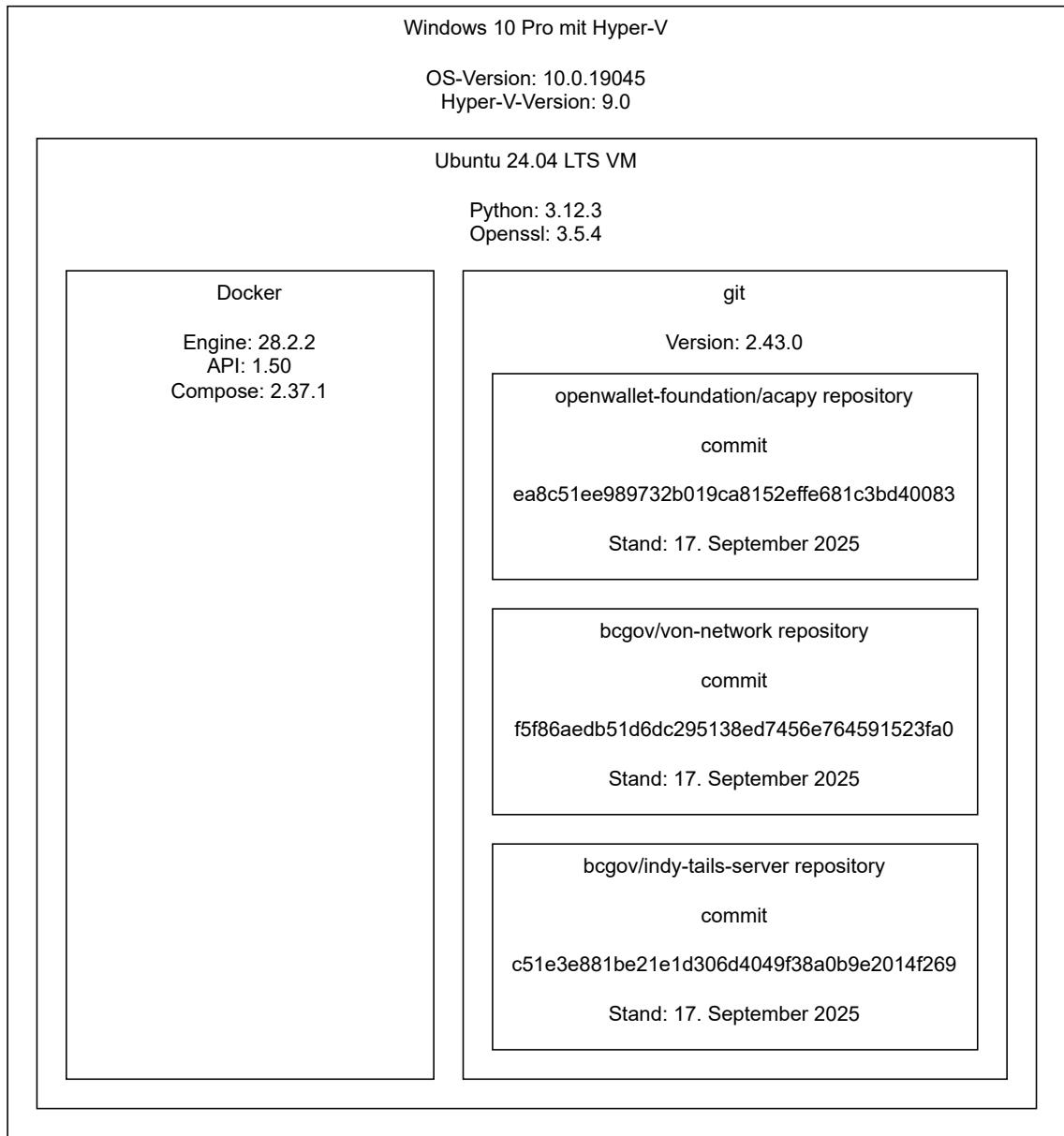
Für die softwareseitige Umsetzung des Sidecar-Proxy-Pattern wird nginx (nginx, n. d.) gewählt, da es mit OpenSSL 3.5+ kompiliert über den OQS-Provider NIST-standardisierte PQC-Algorithmen unterstützt („Building Nginx from Sources“, n. d.).

Anhang 3.2: Implementierung

Anhang 3.2.1: Setup der Entwicklungsumgebung

Die Sicherstellung der wissenschaftlichen Validität von Software-Experimenten erfordert eine robuste Konzeption der zugrundeliegenden Entwicklungsumgebung. Gerade im Kontext komplexer Systemarchitekturen stellt die heterogene und sich rasch wandelnde Natur von Computerumgebungen eine signifikante Hürde für die Reproduzierbarkeit und Erweiterbarkeit von Forschungsergebnissen dar (Boettiger, 2015, S. 1). Um diesen Herausforderungen zu begegnen und die Abhängigkeiten des SSI-Prototypen kontrollierbar zu machen, wurde eine hierarchisch isolierte Umgebung entworfen (Abbildung A-6).

Abbildung A-6
Setup der Entwicklungsumgebung



Anmerkung. Eigene Darstellung.

Host-System und Virtualisierung

Als physische Basis dient eine Workstation, auf der das Betriebssystem Windows 10 Pro (Microsoft, n. d.) (Build 10.0.19045) ausgeführt wird. Um eine hinreichende Isolation zwischen der regulären Arbeitsumgebung zu erreichen, kommt der Typ-1-Hypervisor Hyper-V (Version 9.0) zum Einsatz. Hyper-V wird direkt auf der Computerhardware ausgeführt und liefert robuste Isolation für virtualisierte Workloads (Microsoft, 2025), indem es jedem virtuellen Gastsystem einen separaten Kernel bereitstellt und damit eine stärkere Isolation erreicht als Systeme, die einen gemeinsamen Kernel nutzen (Moore & Zenla, 2025, S. 3).

Innerhalb dieser Virtualisierungsschicht wird Ubuntu 24.04 LTS (Canonical Ltd., 2024) als Gast-Betriebssystem betrieben. Die Wahl einer Linux-Distribution mit Long Term Support ist dadurch begründet, da diese fünf Jahre lang Standard-Support und Sicherheitsupdates bietet, während Zwischenversionen nur neun Monate lang unterstützt werden (Canonical Ltd., 2024, S. 350).

Containerisierung und Orchestrierung

Zur Lösung des in der Softwaretechnik bekannten Problems der Versionsinkompatibilität, bei dem unterschiedliche Versionen von Softwareverzeichnissen die erfolgreiche Ausführung von Code auf fremden Systemen verhindern (Boettiger, 2015, S. 1–2), wird Docker (Engine Version 28.2.2, API 1.50) eingesetzt. Docker ist eine offene Plattform zur Anwendungsentwicklung und zum Betrieb von Anwendungen in isolierten Umgebungen, die es Entwicklern ermöglicht, Anwendungen unabhängig von der zugrundeliegenden Infrastruktur zu verpacken und auszuführen (Docker Inc., n. d. b). Die Containerisierung adressiert die Herausforderung der Reproduzierbarkeit durch die Kapslung des gesamten Softwarestacks in unveränderlichen Images, wobei eine Dockerfile die notwendigen Abhängigkeiten vom Betriebssystem aufwärts definiert (Boettiger, 2015, S. 3–4). Ein Docker-Image, das von einer Dockerfile erstellt wurde, enthält alle Betriebssystem-Bibliotheken, Konfigurationsdateien und Laufzeitumgebungen, welche die Wiederholung der Forschung auf verschiedenen Maschinen konsistent ermöglichen (Boettiger, 2015, S. 4), da Docker das Packaging und die Ausführung eines Containers so handhabt, dass dieser auf verschiedenen Maschinen identisch funktioniert (Docker Inc., n. d. b).

Die Orchestrierung der verteilten Dienste erfolgt mittels Docker Compose (Version 2.37.1). Docker Compose ist ein Werkzeug zur Definition und zum Ausführen von Multi-Container-Anwendungen, das die Kontrolle über den gesamten Anwendungsstack durch die Verwaltung von Services, Netzwerken und Volumes in einer einzigen YAML-Konfigurationsdatei vereinfacht (Docker Inc., n. d. a), wodurch das komplexe Zusammenspiel aus Ledger-Knoten, Agenten und Revocation-Servern mit einem einzigen Befehl deterministisch gestartet werden kann.

Laufzeitumgebung und Versionskontrolle

Die Anwendungslogik der SSI-Agenten und Evaluationsskripte ist in Python 3.12.3 implementiert. Python 3.12 ist eine stabile Hauptversion der Python-Programmiersprache, die sich auf Benutzerfreundlichkeit konzentriert, da f-Strings viele Einschränkungen entfernt wurden und „Did-you-mean“-Vorschläge weiterhin verbessert wurden (Turner, n. d.).

Die Versionierung des Quellcodes erfolgt über Git (Version 2.43.0), ein verteiltes Versionskontrollsystem, das die Verlaufsverfolgung von Änderungen ermöglicht (Github, n. d.). Git bietet Entwicklern die Möglichkeit, die vollständige chronologische Dokumentation sämtlicher Projektänderungen, Entscheidungsprozesse und Fortschrittsmeilensteine zentral einzusehen. Durch den Zugriff auf die Versionshistorie eines Projekts erhalten Entwickler den erforderlichen Kontext, um das System zu verstehen und sachgerecht zu Projektbeiträgen zu beginnen, wodurch eine kontinuierliche und nachvollziehbare Änderungshistorie gewährleistet wird (Github, n. d.).

Um die Stabilität der Umgebung zu gewährleisten, beruhen die zentralen Repositories auf einem festen Commit-Stand vom 17. September 2025 (Abbildung A-6).

Anhang 3.2.2: Zertifikatserstellungsworkflow

Listing A-2 zeigt den vollständigen Workflow der Post-Quantum-Zertifikatserstellung für die Sidecar-Proxies mittels OpenSSL 3.5.4 am Beispiel des Issuers.

Listing A-2

Zertifikatserstellungsworkflow

```

1 # OpenSSL 3.5.4 LTS (version mit PQC support)
2 OpenSSL 3.5.4 30 Sep 2025 (Library: OpenSSL 3.5.4 30 Sep 2025)
3
4 # ML-DSA-87 Private Key (höchste Sicherheit für Root)
5 openssl genpkey -algorithm mldsa87 -out rootCA.key
6
7 # Self-signed Root Certificate (10 Jahre)
8 openssl req -x509 -new -key rootCA.key -out rootCA.crt \
9   -days 3650 -subj "/CN=My PQC Root CA/O=MyOrg/C=DE" \
10  -addext "basicConstraints=critical,CA:TRUE" \
11  -addext "keyUsage=critical,keyCertSign,cRLSign"
12
13 # ML-DSA-65 Private Keys für Sidecar-Proxy
14 openssl genpkey -algorithm mldsa65 -out server.key
15
16 # Certificate Signing Requests (CSRs) für Sidecar-Proxy mit SAN
17 openssl req -new -key server.key -out server.csr \
18   -subj "/CN=server proxy/O=FM/C=DE" \
19   -addext "subjectAltName=DNS:issuer,DNS:pqc-sidecarproxy-issuer,DNS:host.docker.\
internal,DNS:localhost,IP:127.0.0.1"
```

```
20
21 # Signierung mit ML-DSA-65 (Balance zwischen Sicherheit und Performance)
22 openssl x509 -req -in server.csr \
23   -CA rootCA.crt -CAkey rootCA.key -CAcreateserial \
24   -out server.crt -days 365 -sha3-256 \
25   -copy_extensions copy
```

Die Zertifikatsinfrastruktur für die PQC-Sidecar-Proxies basiert auf einer selbstsignierten Root CA, die mit dem Post-Quantum-Signaturalgorithmus ML-DSA-87 erstellt wurde. Dieses Zertifikat besitzt eine Gültigkeit von zehn Jahren und wird in den System-Trust-Store aller Container importiert, sodass Python-Bibliotheken und Kommandozeilen-Tools die ML-DSA-65-signierten Zertifikate automatisch als vertrauenswürdig erkennen.

Für jeden der fünf Sidecar-Proxies wurde ein dediziertes Server-Zertifikat generiert: issuer.crt, holder.crt, verifier.crt, von-webserver.crt und tails-server.crt. Jedes dieser Zertifikate wird mittels Certificate Signing Request erstellt und von der Root CA mit ML-DSA-65 signiert. Die Zertifikate enthalten Subject Alternative Names, die sowohl den DNS-Namen „localhost“ als auch die IP-Adresse „127.0.0.1“ umfassen, um flexible Zugriffsmöglichkeiten während der Entwicklung zu ermöglichen.

Die Integration der Zertifikate erfolgt über Docker-Volume-Mounts. Das Verzeichnis „/opt/pqc_sidecarproxy_nginx/certs/“ wird als Read-Only-Volume in jeden nginx-Container unter „/opt/nginx/certs/“ eingebunden. In der nginx-Konfiguration werden die Zertifikate durch die Direktiven „ssl_certificate /opt/nginx/certs/<agent>.crt“ und „ssl_certificate_key /opt/nginx/certs/<agent>.key“ referenziert. Parallel dazu wird das Root CA-Zertifikat in allen ACA-Py-Agent-Containern nach „/usr/local/share/ca-certificates/pqc-root-ca.crt“ kopiert und mittels „update-ca-certificates“ in den System-Trust-Store importiert. Zusätzlich werden die Umgebungsvariablen „SSL_CERT_FILE“, „REQUESTS_CA_BUNDLE“ und „CURL_CA_BUNDLE“ auf „/etc/ssl/certs/ca-certificates.crt“ gesetzt, um sicherzustellen, dass Python-Anwendungen und HTTP-Clients die importierte Root CA verwenden und TLS-Verbindungen zu den PQC-Sidecar-Proxies erfolgreich validieren können.

Anhang 3.2.3: Dockerfile: Sidecar-Proxy nginx

Listing A-3 beinhaltet den Code des Dockerfiles parallel zur visuellen Darstellung in Abbildung 13.

In der Build-Phase findet die gesamte Kompilierung und Zusammenführung der für den Betrieb erforderlichen Abhängigkeiten statt. Zunächst werden Versionsangaben für Bibliotheken, Algorithmen, TLS-Gruppen und Installationsverzeichnisse als Build-Parameter definiert. Anschließend werden die projektrelevanten Quellen wie liboqs, oqs-provider, openssl, nginx und curl gezielt und versioniert aus ihren jeweiligen Repositories geladen. Die Kompilierung erfolgt dabei in einer spezifischen Reihenfolge. OpenSSL und liboqs werden mit definierten Einstellungen gebaut, nginx wird unter Einbindung des oqs-provider und einer hybrid gelinkten OpenSSL-Version kompiliert, und curl wird dynamisch mit OpenSSL gebaut, um HTTPS-Unterstützung sicherzustellen. Abschließend werden die generierten Binaries optimiert und nicht benötigte Dateien entfernt, um das finale Image so schlank wie möglich zu gestalten.

Die Runtime-Stage bildet daraufhin das minimalistische Endprodukt, das zum produktiven Einsatz bestimmt ist. Die in der Build-Phase erzeugten Binaries und Konfigurationen werden hier in das neue Runtime-Image übernommen. Die notwendigen Einstellungen für Post-Quantum-TLS-Gruppen und Logging werden über Umgebungsvariablen und angepasste Konfigurationen gesetzt. Das Image läuft nach dem Least-Privilege-Prinzip (Khattar, 2025, S. 1) mit einem dedizierten Non-Root User (oqs), und der Container-Entrypoint ist klar über den Aufruf von nginx definiert.

Gegenüber dem Original-Dockerfile (open-quantum-safe, 2025) wurden mehrere zielgerichtete Modifikationen (in Abbildung 13 in rot markiert) vorgenommen. Die OpenSSL-Version wurde von variablen Tags auf die explizite Versionsnummer „3.5.4“ festgelegt. Die für den Key-Encapsulation-Mechanism zentralen TLS-Gruppen wurden auf X25519MLKEM768, mlkem768, x25519 und mlkem1024 reduziert, um gezielt ML-KEM-768 zu forcieren. Im Originalskript werden Zertifikate direkt während des Builds generiert. Die modifizierte Version hingegen verzichtet darauf und sieht vor, dass Zertifikate via Volume von außen eingebracht werden. Die Build-Reihenfolge wurde neu strukturiert. OpenSSL wird zuerst als Shared Library kompiliert, um eine dynamische Verlinkung von curl und nginx zu ermöglichen. Dies umgeht auch Inkompatibilitäten zwischen Alpine Linux und statisch gelinkten OpenSSL-Versionen. Die Optimierung und Entfernung nicht benötigter Artefakte erfolgt granularer, wobei dynamische Binaries und Modul-Dateien wie oqsprovider.so gezielt gestreift werden. Das finale Runtime-Image wurde zusätzlich verschlankt: Es werden ca-certificates als Runtime-Abhängigkeit hinzugefügt, Ports werden explizit nicht

mehr im Dockerfile exponiert, dies erfolgt stattdessen via Docker Compose und der ENTRYPOINT ist strikt auf den Sidecar-Proxy-Einsatz ausgerichtet.

Listing A-3

Dockerfile - Sidecar-Proxy nginx

```

1 # Nginx with OpenSSL 3.5.4 (LTS) + OQS Provider for Post-Quantum Cryptography
2 # Based on: https://github.com/open-quantum-safe/oqs-demos/blob/main/nginx/Dockerfile
3 #
4 # Customizations:
5 # - OpenSSL 3.5.4 (LTS with Security Fixes)
6 # - Uses custom certificates (mounted via volume)
7 # - ML-KEM-768 Key Exchange enabled
8
9 # Define build arguments for version tags, installation paths, and configurations
10 ARG ALPINE_VERSION=3.21
11 ARG OPENSSL_TAG=openssl-3.5.4
12 ARG LIBOQS_TAG=0.13.0
13 ARG OQSPROVIDER_TAG=0.9.0
14 ARG NGINX_VERSION=1.28.0
15 ARG BASEDIR="/opt"
16 ARG INSTALLDIR=${BASEDIR}/nginx
17
18 # Specify supported signature and key encapsulation mechanisms (KEM) algorithms
19 ARG SIG_ALG="mldsa65"
20 ARG DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
21
22 # Stage 1: Build - Compile and assemble all necessary components and dependencies
23 FROM alpine:${ALPINE_VERSION} AS intermediate
24 ARG OPENSSL_TAG
25 ARG LIBOQS_TAG
26 ARG OQSPROVIDER_TAG
27 ARG NGINX_VERSION
28 ARG BASEDIR
29 ARG INSTALLDIR
30 ARG SIG_ALG
31 ARG DEFAULT_GROUPS
32 ARG OSSLDIR=${BASEDIR}/openssl/.openssl
33
34 # Install required build tools and system dependencies
35 RUN apk update && apk --no-cache add \
36     build-base linux-headers libtool \
37     automake autoconf make cmake ninja \
38     openssl openssl-dev git wget pcre-dev
39
40 # Download and prepare source files needed for the build process
41 WORKDIR /opt
42 RUN git clone --depth 1 --branch ${LIBOQS_TAG} https://github.com/open-quantum-safe/
        liboqs \
43     && git clone --depth 1 --branch ${OQSPROVIDER_TAG} https://github.com/open-quantum-
        -safe/oqs-provider.git \
44     && git clone --depth 1 --branch ${OPENSSL_TAG} https://github.com/openssl/openssl.
        git \
45     && wget -q nginx.org/download/nginx-${NGINX_VERSION}.tar.gz \
46     && tar -zxf nginx-${NGINX_VERSION}.tar.gz \
47     && rm nginx-${NGINX_VERSION}.tar.gz

```

```

48
49 # Build and install OpenSSL with shared libraries (for curl and nginx)
50 WORKDIR /opt/openssl
51 RUN ./Configure --prefix=${OSSLDIR} \
52     --openssldir=${OSSLDIR}/ssl \
53     shared \
54     enable-fips \
55     && make -j"${nproc}" \
56     && make install_sw install_ssldirs
57
58 # Configure OpenSSL to support the oqs-provider
59 RUN cp /opt/openssl/apps/openssl.cnf ${OSSLDIR}/ssl/ && \
60     sed -i "s/default = default_sect/default = default_sect\nnoqsp provider = \
61     oqsprovider_sect/g" ${OSSLDIR}/ssl/openssl.cnf && \
62     sed -i "s/\\[default_sect\\]\\/[default_sect\\]\\nactivate = 1\\n\\[oqsprovider_sect\\]\\\
63     nactivate = 1\\n/g" ${OSSLDIR}/ssl/openssl.cnf && \
64     sed -i "s/providers = provider_sect/providers = provider_sect\\nssl_conf = ssl_sect \
65     \\n\\n\\[ssl_sect\\]\\nsystem_default = system_default_sect\\n\\n\\[ \
66     system_default_sect\\]\\nGroups = \$ENV\\:\\:DEFAULT_GROUPS\\n/g" ${OSSLDIR}/ssl/ \
67     openssl.cnf && \
68     sed -i "s/HOME\\t\\t\\t= ./HOME\\t\\t\\t= .\\nDEFAULT_GROUPS\\t= ${DEFAULT_GROUPS}\\t= ${DEFAULT_GROUPS}/g" ${
69     OSSLDIR}/ssl/openssl.cnf
70
71 # Build and install liboqs
72 WORKDIR /opt/liboqs/build
73 RUN cmake -G"Ninja" \
74     -DOQS_DIST_BUILD=ON \
75     -DBUILD_SHARED_LIBS=OFF \
76     -DCMAKE_INSTALL_PREFIX="${INSTALLDIR}" .. \
77     && ninja -j"${nproc}" && ninja install
78
79 # Build and install Nginx with shared OpenSSL
80 WORKDIR /opt/nginx-${NGINX_VERSION}
81 RUN ./configure --prefix=${INSTALLDIR} \
82     --with-debug --with-http_ssl_module \
83     --with-cc-opt="-I${OSSLDIR}/include" \
84     --with-ld-opt="--L${OSSLDIR}/lib64 -Wl,-rpath,${OSSLDIR}/lib64" \
85     --without-http_gzip_module && \
86     make -j"${nproc}" && make install
87
88 # Build and install OQS provider
89 WORKDIR /opt/oqs-provider
90 RUN ln -s "/opt/nginx/include/oqs" "${OSSLDIR}/include" && \
91     rm -rf build && \
92     cmake -DCMAKE_BUILD_TYPE=Debug \
93         -DOPENSSL_ROOT_DIR="${OSSLDIR}" \
94         -DCMAKE_PREFIX_PATH="${INSTALLDIR}" \
95         -S . -B build && \
96     cmake --build build && \
97     MODULESDIR=$(find "${OSSLDIR}" -name ossl-modules -type d | head -1) && \
98     export MODULESDIR && \
99     cp build/lib/oqsprovider.so "${MODULESDIR}" && \
100    rm -rf "${INSTALLDIR:?}/lib64"
101
102 # Build curl with shared OpenSSL 3.5.4
103 ARG CURL_VERSION=8.11.1
104 WORKDIR /opt

```

```

99 RUN wget -q https://curl.se/download/curl-${CURL_VERSION}.tar.gz && \
100 tar -zxf curl-${CURL_VERSION}.tar.gz && \
101 rm curl-${CURL_VERSION}.tar.gz
102
103 WORKDIR /opt/curl-${CURL_VERSION}
104 RUN LDFLAGS="-Wl,-rpath,${OSSLDIR}/lib64 -L${OSSLDIR}/lib64" \
105 PKG_CONFIG_PATH="${OSSLDIR}/lib64/pkgconfig" \
106 ./configure \
107 --prefix=${INSTALLDIR} \
108 --with-openssl=${OSSLDIR} \
109 --with-ca-bundle=/etc/ssl/certs/ca-certificates.crt \
110 --disable-manual \
111 --disable-ldap \
112 --disable-ldaps \
113 --without-libpsl \
114 --without-zlib \
115 --without-brotli \
116 --without-zstd && \
117 make -j"${nproc}" && \
118 make install
119
120 # Minimize image size by stripping binaries
121 WORKDIR ${INSTALLDIR}
122 ENV PATH="${INSTALLDIR}/sbin:${INSTALLDIR}/bin:${OSSLDIR}/bin:${PATH}"
123
124 RUN set -ex && \
125 strip "${OSSLDIR}/lib64/*.a \
126     "${OSSLDIR}/lib64/openssl-modules/oqsprovider.so" \
127     "${INSTALLDIR}/sbin/* \
128     "${INSTALLDIR}/bin/curl" \
129     "${OSSLDIR}/bin/openssl" && \
130 mkdir -p certs
131
132 # Stage 2: Runtime - Create a lightweight image with essential binaries and
133 # configurations
134 FROM alpine:${ALPINE_VERSION}
135 ARG INSTALLDIR
136 ARG BASEDIR
137 ARG OSSLDIR=${BASEDIR}/openssl/.openssl
138
139 # Install runtime dependencies
140 RUN apk update && apk --no-cache add pcre-dev ca-certificates
141
142 # Copy compiled artifacts and configuration from the intermediate stage
143 COPY --from=intermediate ${INSTALLDIR} ${INSTALLDIR}
144 COPY --from=intermediate ${OSSLDIR} ${OSSLDIR}
145
146 # Link logs to Docker collector
147 RUN set -ex && \
148     mkdir -p "${INSTALLDIR}/logs" && \
149     ln -sf /dev/stdout "${INSTALLDIR}/logs/access.log" && \
150     ln -sf /dev/stderr "${INSTALLDIR}/logs/error.log"
151
152 # Expose HTTPS port
153 # EXPOSE 443
154 # Set OpenSSL configuration environment

```

```

155 # From Nginx 1.25.2: "nginx does not try to load OpenSSL configuration if the
156 # --with-openssl option was used to build OpenSSL and the OPENSSL_CONF
157 # environment variable is not set." Hence we must explicitly set OPENSSL_CONF.
158 ENV PATH="${INSTALLDIR}/sbin:${INSTALLDIR}/bin:${OSSLDIR}/bin:${PATH}" \
159   OPENSSL_CONF="${OSSLDIR}/ssl/openssl.cnf" \
160   DEFAULT_GROUPS="X25519MLKEM768:mlkem768:x25519:mlkem1024"
161
162 # Create non-root user and update permissions
163 RUN addgroup -g 1000 -S oqs \
164   && adduser --uid 1000 -S oqs -G oqs \
165   && chown -R oqs:oqs "${INSTALLDIR}"
166
167 # Run as non-root user
168 USER oqs
169 WORKDIR ${INSTALLDIR}
170
171 STOPSIGNAL SIGTERM
172 CMD ["nginx", "-c", "nginx-conf/nginx.conf", "-g", "daemon off;"]

```

Anhang 3.2.4: nginx_holder.conf

Strukturell folgen alle nginx-Konfigurationen einem konsistenten Schichtenmodell. Die globale Konfigurationsebene definiert Worker-Prozesse, Logging-Parameter sowie HTTP-Grundeinstellungen. Die mittlere Ebene besteht aus Upstream-Blöcken, die interne Services abstrahieren. Die oberste Ebene enthält Server-Blöcke, die für jeden öffentlich erreichbaren Endpoint eine HTTPS-Schnittstelle exponieren.

Das zentrale Sicherheitsmerkmal aller Konfigurationen ist die Direktive „ssl_ecdh_curve X25519MLKEM768“, die den Hybrid-Key-Exchange zwischen klassischer Elliptischen-Kurven-Kryptografie und dem post-quantensicheren ML-KEM-768-Algorithmus aktiviert. Die konkrete Gruppenauswahl wird extern über die Umgebungsvariable „DEFAULT_GROUPS“ gesteuert, was eine Trennung von Konfiguration und Deploymentslogik ermöglicht. Komplementär hierzu werden SSL-Zertifikate mit ML-DSA-65-Signaturalgorithmus verwendet, die als Volume-Mounts unter „/opt/nginx/certs/“ eingebunden werden. Dieser Ansatz gewährleistet, dass sowohl der Schlüsselaustausch als auch die Server-Authentifikation post-quantensicher erfolgen, während die Zertifikate unabhängig von der Container-Runtime verwaltet und rotiert werden können.

Die Reverse-Proxy-Funktionalität wird durch „location /“-Blöcke realisiert, die Anfragen an Upstream-Services weiterleiten. Alle Konfigurationen exponieren einen „/health“-Endpoint für Orchestrierungs-Health-Checks.

Listing A-4*nginx_holder.conf*

```

1 # OQS Nginx Configuration for VON Network Webserver Reverse Proxy
2 # Post-Quantum Cryptography enabled with ML-KEM
3
4 worker_processes auto;
5 error_log /opt/nginx/logs/error.log info;
6 pid /opt/nginx/logs/nginx.pid;
7
8 events {
9     worker_connections 1024;
10 }
11
12 http {
13     include /opt/nginx/conf/mime.types;
14     default_type application/octet-stream;
15
16     log_format main '$remote_addr - $remote_user [$time_local] "$request" '
17             '$status $body_bytes_sent "$http_referer" '
18             '"$http_user_agent" "$http_x_forwarded_for"';
19
20     access_log /opt/nginx/logs/access.log main;
21
22     sendfile on;
23     keepalive_timeout 65;
24
25     # Upstream: Holder Agent Inbound Transport (Port 8030)
26     upstream holder_inbound {
27         server holder:8030;
28     }
29
30     # Upstream: Holder Agent Admin API (Port 8031)
31     upstream holder_admin {
32         server holder:8031;
33     }
34
35     # HTTPS Server for Holder Inbound Transport (Port 8030)
36     server {
37         listen 8030 ssl;
38         server_name pqc-sidecarproxy-holder;
39
40         # SSL Certificates (custom ML-DSA-65 certificates)
41         ssl_certificate /opt/nginx/certs/holder.crt;
42         ssl_certificate_key /opt/nginx/certs/holder.key;
43
44         # TLS 1.3 with Post-Quantum Cryptography
45         ssl_protocols TLSv1.3;
46         ssl_ecdh_curve X25519MLKEM768:x25519;
47         # Quantum-Safe Key Exchange Groups (ML-KEM from NIST FIPS-203)
48         # Groups are set via DEFAULT_GROUPS environment variable
49         # Default: mlkem768:x25519:mlkem1024
50         # TLS 1.3 cipher suites are automatically selected
51
52         ssl_prefer_server_ciphers off;
53
54         # Reverse Proxy to Holder Inbound Transport

```

```

55     location / {
56         proxy_pass http://holder_inbound;
57         proxy_set_header Host $host;
58         proxy_set_header X-Real-IP $remote_addr;
59         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
60         proxy_set_header X-Forwarded-Proto https;
61
62         # Timeouts
63         proxy_connect_timeout 60s;
64         proxy_send_timeout 60s;
65         proxy_read_timeout 60s;
66     }
67 }
68
69 # HTTPS Server for Holder Admin API (Port 8031)
70 server {
71     listen 8031 ssl;
72     server_name pqc-sidecarproxy-holder-admin;
73
74     # SSL Certificates (ML-DSA-65)
75     ssl_certificate /opt/nginx/certs/holder.crt;
76     ssl_certificate_key /opt/nginx/certs/holder.key;
77
78     # TLS 1.3 with Post-Quantum Cryptography
79     ssl_protocols TLSv1.3;
80     ssl_ecdh_curve X25519MLKEM768:x25519;
81     # Quantum-Safe Key Exchange Groups (ML-KEM from NIST FIPS-203)
82     # Groups are set via DEFAULT_GROUPS environment variable
83     # Default: X25519MLKEM768:mlkem768:x25519:mlkem1024
84
85     ssl_prefer_server_ciphers off;
86
87     # Reverse Proxy to Holder Admin API
88     location / {
89         proxy_pass http://holder_admin;
90         proxy_set_header Host $host;
91         proxy_set_header X-Real-IP $remote_addr;
92         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
93         proxy_set_header X-Forwarded-Proto https;
94
95         # Timeouts
96         proxy_connect_timeout 60s;
97         proxy_send_timeout 60s;
98         proxy_read_timeout 60s;
99     }
100
101    # Health Check
102    location /health {
103        access_log off;
104        return 200 "Holder Admin PQC Proxy OK\n";
105        add_header Content-Type text/plain;
106    }
107 }
108 }
```

Anhang 3.2.5: DLT-Infrastruktur

Die unmodifizierte von-network-Implementierung stellt vier Indy-Validator-Nodes (bcgov, n. d. g) bereit, die über das „Practical Byzantine Fault Tolerance“-Konsensprotokoll synchronisiert werden (hyperledger, n. d.). Jeder Node exponiert zwei Ports für die Indy-Protokollkommunikation (9701–9708) und verfügt über ein dediziertes Docker-Volume zur Persistierung des Ledger-Zustands (bcgov, n. d. f). Zusätzlich zu den Validator-Nodes wird ein Webserver-Container deployt, der ein Web-Interface zur Ledger-Visualisierung sowie einen HTTP-Endpoint zur Genesis-File-Distribution bereitstellt (bcgov, n. d. f). Diese Architektur ermöglicht es ACA-Py-Agents, beim Start die Genesis-Transaktionsdatei über eine konfigurierbare URL abzurufen und sich automatisch mit dem Indy-Netzwerk zu verbinden.

Neben der Proxy-Integration wurden die Indy-Validator-Nodes und der Webserver-Container vollständig unverändert aus dem Original-Repository übernommen. Dies umfasst das Base-Image „ghcr.io/bcgov/von-image:node-1.12-6“ (bcgov, n. d. e), die Genesis-Generierungslogik in „scripts/init_genesis.sh“ (bcgov, n. d. i) sowie die Node-Startup-Scripts „scripts/start_node.sh“ (bcgov, n. d. i). Die Beibehaltung des ursprünglichen Indy-Node-Codes gewährleistet, dass die blockchain-interne Validierung, Konsens-Mechanismen und Ledger-Operationen identisch mit etablierten Indy-Deployments funktionieren und ausschließlich die externe Kommunikationsschicht durch PQC abgesichert wird.

Die Ledger-Struktur von Hyperledger Indy bleibt ebenfalls unverändert und umfasst drei logische Ledger-Typen. Der Domain Ledger speichert DIDs, Schemas und Credential Definitions, der Pool Ledger verwaltet die Validator-Node-Registry, der Config Ledger enthält Netzwerk-Konfigurationen wie Transaction Author Agreements und Acceptable Mechanism Lists (bcgov, n. d. h). Die Persistierung dieser Ledger erfolgt über Docker-Volumes (node1-data bis node4-data) (bcgov, n. d. f).

Listing A-5

docker-compose.yml: DLT-Infrastruktur

```

1 version: '3'
2 services:
3   #
4   # Client
5   #
6   client:
7     image: von-network-base
8     command: ./scripts/start_client.sh
9     environment:
10    - IP=${IP}
11    - IPS=${IPS}
12    - DOCKERHOST=${DOCKERHOST}
```

```

13      - RUST_LOG=${RUST_LOG}
14 networks:
15   - von
16 volumes:
17   - client-data:/home/indy/.indy_client
18   - ./tmp:/tmp
19
20 #
21 # Webserver
22 #
23 webserver:
24   image: von-network-base
25   command: bash -c 'sleep 10 && ./scripts/start_webserver.sh'
26   container_name: von-webserver
27   environment:
28     - IP=${IP}
29     - IPS=${IPS}
30     - DOCKERHOST=${DOCKERHOST}
31     - LOG_LEVEL=${LOG_LEVEL}
32     - RUST_LOG=${RUST_LOG}
33     - GENESIS_URL=${GENESIS_URL}
34     - LEDGER_SEED=${LEDGER_SEED}
35     - LEDGER_CACHE_PATH=${LEDGER_CACHE_PATH}
36     - MAX_FETCH=${MAX_FETCH:-50000}
37     - RESYNC_TIME=${RESYNC_TIME:-120}
38     - POOL_CONNECTION_ATTEMPTS=${POOL_CONNECTION_ATTEMPTS:-5}
39     - POOL_CONNECTION_DELAY=${POOL_CONNECTION_DELAY:-10}
40     - REGISTER_NEW_DIDS=${REGISTER_NEW_DIDS:-True}
41     - ENABLE_LEDGER_CACHE=${ENABLE_LEDGER_CACHE:-True}
42     - ENABLE_BROWSER_ROUTES=${ENABLE_BROWSER_ROUTES:-True}
43     - DISPLAY_LEDGER_STATE=${DISPLAY_LEDGER_STATE:-True}
44     - LEDGER_INSTANCE_NAME=${LEDGER_INSTANCE_NAME:-localhost}
45     - LEDGER_DESCRIPTION=${LEDGER_DESCRIPTION}
46     - WEB_ANALYTICS_SCRIPT=${WEB_ANALYTICS_SCRIPT}
47     - INFO_SITE_TEXT=${INFO_SITE_TEXT}
48     - INFO_SITE_URL=${INFO_SITE_URL}
49     - INDY_SCAN_URL=${INDY_SCAN_URL}
50     - INDY_SCAN_TEXT=${INDY_SCAN_TEXT}
51 networks:
52   - von
53 #
54 # ports:
55 #   - ${WEB_SERVER_HOST_PORT:-9000}:8000
56 volumes:
57   - ./config:/home/indy/config
58   - ./server:/home/indy/server
59   - webserver-cli:/home/indy/.indy-cli
60   - webserver-ledger:/home/indy/ledger
61
62 #
63 # Synchronization test
64 #
65 synctest:
66   image: von-network-base
67   command: ./scripts/start_synctest.sh
68   environment:
69     - IP=${IP}
     - IPS=${IPS}

```

```

70      - DOCKERHOST=${DOCKERHOST}
71      - LOG_LEVEL=${LOG_LEVEL}
72      - RUST_LOG=${RUST_LOG}
73 networks:
74     - von
75 ports:
76     - ${WEB_SERVER_HOST_PORT:-9000}:8000
77 volumes:
78     - ./config:/home/indy/config
79     - ./server:/home/indy/server
80     - webserver-cli:/home/indy/.indy-cli
81     - webserver-ledger:/home/indy/ledger
82
83 #
84 # Nodes
85 #
86 nodes:
87   image: von-network-base
88   command: ./scripts/start_nodes.sh
89   networks:
90     - von
91   ports:
92     - 9701:9701
93     - 9702:9702
94     - 9703:9703
95     - 9704:9704
96     - 9705:9705
97     - 9706:9706
98     - 9707:9707
99     - 9708:9708
100  environment:
101    - IP=${IP}
102    - IPS=${IPS}
103    - DOCKERHOST=${DOCKERHOST}
104    - LOG_LEVEL=${LOG_LEVEL}
105    - RUST_LOG=${RUST_LOG}
106  volumes:
107    - nodes-data:/home/indy/ledger
108
109 node1:
110   image: von-network-base
111   command: ./scripts/start_node.sh 1
112   networks:
113     - von
114   ports:
115     - 9701:9701
116     - 9702:9702
117   container_name: von-node1
118   environment:
119     - IP=${IP}
120     - IPS=${IPS}
121     - DOCKERHOST=${DOCKERHOST}
122     - LOG_LEVEL=${LOG_LEVEL}
123     - RUST_LOG=${RUST_LOG}
124   volumes:
125     - node1-data:/home/indy/ledger
126

```

```

127  node2:
128    image: von-network-base
129    command: ./scripts/start_node.sh 2
130    networks:
131      - von
132    ports:
133      - 9703:9703
134      - 9704:9704
135    container_name: von-node2
136    environment:
137      - IP=${IP}
138      - IPS=${IPS}
139      - DOCKERHOST=${DOCKERHOST}
140      - LOG_LEVEL=${LOG_LEVEL}
141      - RUST_LOG=${RUST_LOG}
142    volumes:
143      - node2-data:/home/indy/ledger
144
145  node3:
146    image: von-network-base
147    command: ./scripts/start_node.sh 3
148    networks:
149      - von
150    ports:
151      - 9705:9705
152      - 9706:9706
153    container_name: von-node3
154    environment:
155      - IP=${IP}
156      - IPS=${IPS}
157      - DOCKERHOST=${DOCKERHOST}
158      - LOG_LEVEL=${LOG_LEVEL}
159      - RUST_LOG=${RUST_LOG}
160    volumes:
161      - node3-data:/home/indy/ledger
162
163  node4:
164    image: von-network-base
165    command: ./scripts/start_node.sh 4
166    networks:
167      - von
168    ports:
169      - 9707:9707
170      - 9708:9708
171    container_name: von-node4
172    environment:
173      - IP=${IP}
174      - IPS=${IPS}
175      - DOCKERHOST=${DOCKERHOST}
176      - LOG_LEVEL=${LOG_LEVEL}
177      - RUST_LOG=${RUST_LOG}
178    volumes:
179      - node4-data:/home/indy/ledger
180
181  # Post-Quantum Nginx Reverse Proxy für VON Network Webserver
182  pqc-sidecarproxy-webserver:
183    build:

```

```

184     context: ./pqc_sidecarproxy_nginx
185     dockerfile: Dockerfile
186     args:
187       OPENSSL_TAG: openssl-3.5.4
188       LIBOQS_TAG: 0.13.0
189       OQSPROVIDER_TAG: 0.9.0
190       NGINX_VERSION: 1.28.0
191       SIG_ALG: mldsa65
192       DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
193     container_name: von-pqc-sidecarproxy-webserver
194     environment:
195       # OpenSSL Configuration
196       - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
197       # Post-Quantum Key Exchange Groups
198       - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
199     networks:
200       - von
201       - sidecarproxy
202     ports:
203       - 8000:8000 # HTTPS with Post-Quantum Cryptography (ML-KEM-768)
204     volumes:
205       # Custom nginx configuration for reverse proxy
206       - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_webserver.conf:/opt/nginx/nginx-conf
207           /nginx.conf:ro
208       # Custom ML-DSA-65 certificates
209       - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
210       # Logs
211       - nginx-logs:/opt/nginx/logs
212     depends_on:
213       - webserver
214     restart: unless-stopped
215     healthcheck:
216       test: ["CMD", "curl", "-k", "-f", "https://localhost:8000/health"]
217       interval: 30s
218       timeout: 10s
219       retries: 5
220       start_period: 10s
221     networks:
222       von:
223       sidecarproxy:
224
225     volumes:
226       client-data:
227       webserver-cli:
228       webserver-ledger:
229       node1-data:
230       node2-data:
231       node3-data:
232       node4-data:
233       nodes-data:
234       nginx-logs:

```

Anhang 3.2.6: Revocation Registry

Der unmodifizierte indy-tails-server wird durch das Skript „./manage start“ (bcgov, n. d. c) aus dem „docker“-Verzeichnis deployt, welches als Wrapper um Docker-Compose-Befehle fungiert und die Orchestrierung mehrerer Container-Services koordiniert. Der indy-tails-server läuft standardmäßig auf Port 6543 (bcgov, n. d. d).

Der Server exponiert vier REST-Endpunkte für Upload- und Download-Operationen, wobei zwei Adressierungsmethoden unterstützt werden. Für Upload-Operationen steht der Endpunkt „PUT /revoc_reg_id“ zur Verfügung, der als Multipart-File-Upload mit zwei Pflichtfeldern implementiert ist. Das erste Feld muss „genesis“ (Genesis-Transaktionsdatei) heißen, das zweite „tails“ (Tails-Datei) (bcgov, n. d. d). Der Server validiert die Integrität der hochgeladenen Datei, indem er die Revocation Registry Definition aus dem Ledger abruft und den Hash mit dem „fileHash“-Attribut abgleicht (bcgov, n. d. d). Als Alternative existiert der Hash-basierte Upload-Endpunkt „PUT /hash/tails-hash“, der die Datei gegen den übermittelten Hash validiert und zusätzliche Struktur-Checks durchführt (bcgov, n. d. d). Die Download-Endpunkte folgen einem analogen Schema mit „GET /revoc_reg_id“ und „GET /hash/tails-hash“ für den Abruf existierender Tails-Dateien (bcgov, n. d. d).

Listing A-6

docker-compose.yml: Revocation Registry

```

1 services:
2   ngrok-tails-server:
3     image: ngrok/ngrok
4     networks:
5       - tails-server
6     ports:
7       - 4044:4040
8     command: start --all
9     environment:
10      - NGROK_CONFIG=/etc/ngrok.yml
11      - NGROK_AUTHTOKEN=${NGROK_AUTHTOKEN}
12     volumes:
13       - ./ngrok.yml:/etc/ngrok.yml
14   tails-server:
15     build:
16       context: ..
17       dockerfile: docker/Dockerfile.tails-server
18     networks:
19       - tails-server
20     command: >
21       tails-server
22         --host 0.0.0.0
23         --port 6543
24         --storage-path $STORAGE_PATH
25         --log-level $LOG_LEVEL
26         --log-config $LOGGING_CONFIG
27   tester:

```

```

28     build:
29         context: ..
30         dockerfile: docker/Dockerfile.test
31
32 # Post-Quantum Nginx Reverse Proxy für Tails Server
33 pqc-sidecarproxy-tails-server:
34     build:
35         context: ./pqc_sidecarproxy_nginx
36         dockerfile: Dockerfile
37     args:
38         OPENSSL_TAG: openssl-3.5.4
39         LIBOQS_TAG: 0.13.0
40         OQSPROVIDER_TAG: 0.9.0
41         NGINX_VERSION: 1.28.0
42         SIG_ALG: mldsa65
43         DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
44     container_name: pqc-sidecarproxy-tails-server
45     environment:
46         # OpenSSL Configuration
47         - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
48         # Post-Quantum Key Exchange Groups
49         - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
50     networks:
51         - tails-server
52         - von_sidecarproxy
53     ports:
54         - 6543:6543 # HTTPS with Post-Quantum Cryptography (ML-KEM-768)
55     volumes:
56         # Custom nginx configuration for reverse proxy
57         - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_tails-server.conf:/opt/nginx/nginx-
58             conf/nginx.conf:ro
59         # Custom ML-DSA-65 certificates
60         - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
61         # Logs
62         - nginx-logs:/opt/nginx/logs
63     depends_on:
64         - tails-server
65     restart: unless-stopped
66     healthcheck:
67         test: ["CMD", "curl", "-k", "-f", "https://localhost:6543/health"]
68         interval: 30s
69         timeout: 10s
70         retries: 5
71         start_period: 10s
72     networks:
73         tails-server:
74         von_sidecarproxy:
75             external: true
76
77     volumes:
78         nginx-logs:

```

Anhang 3.2.7: Dockerfile: acapy-base

Listing A-7

Dockerfile - acapy-base

```

1 ARG python_version=3.12
2 FROM python:${python_version}-slim-bookworm AS build
3
4 RUN pip install --no-cache-dir poetry==2.1.1
5
6 WORKDIR /src
7
8 COPY ./pyproject.toml ./poetry.lock ./
9 RUN poetry install --no-root
10
11 COPY ./acapy_agent ./acapy_agent
12 COPY ./README.md /src
13 RUN poetry build
14
15 FROM python:${python_version}-slim-bookworm AS main
16
17 ARG uid=1001
18 ARG user=aries
19 ARG acapy_name="acapy-agent"
20 ARG acapy_version
21 ARG acapy_reqs=[didcommv2]
22
23 ENV HOME="/home/$user" \
24     APP_ROOT="/home/$user" \
25     LC_ALL=C.UTF-8 \
26     LANG=C.UTF-8 \
27     PIP_NO_CACHE_DIR=off \
28     PYTHONUNBUFFERED=1 \
29     PYTHONNIOENCODING=UTF-8 \
30     RUST_LOG=warn \
31     SHELL=/bin/bash \
32     SUMMARY="$acapy_name image" \
33     DESCRIPTION="$acapy_name provides a base image for running acapy agents in Docker."
34     \
35     This image layers the python implementation of $acapy_name $acapy_version. Based
36     on Debian Buster.
37
38 LABEL summary="$SUMMARY" \
39     description="$DESCRIPTION" \
40     io.k8s.description="$DESCRIPTION" \
41     io.k8s.display-name="$acapy_name $acapy_version" \
42     name=$acapy_name \
43     acapy.version="$acapy_version" \
44     maintainer=""
45
46
47 # Add aries user
48 RUN useradd -U -ms /bin/bash -u $uid $user
49
50
51 # Install environment
52 RUN apt-get update && \
53     apt-get install -y --no-install-recommends \

```

```

50 apt-transport-https \
51 ca-certificates \
52 curl \
53 git \
54 libffi-dev \
55 libgmp10 \
56 libncurses5 \
57 libncursesw5 \
58 openssl \
59 sqlite3 \
60 zlib1g && \
61 apt-get autoremove -y && \
62 apt-get clean -y && \
63 rm -rf /var/lib/apt/lists/* /usr/share/doc/*
64
65 WORKDIR $HOME
66
67 # Add local binaries and aliases to path
68 ENV PATH="$HOME/.local/bin:$PATH"
69
70 # - In order to drop the root user, we have to make some directories writable
71 #   to the root group as OpenShift default security model is to run the container
72 #   under random UID.
73 RUN usermod -a -G 0 $user
74
75 # Create standard directories to allow volume mounting and set permissions
76 # Note: PIP_NO_CACHE_DIR environment variable should be cleared to allow caching
77 RUN mkdir -p \
78     $HOME/.acapy_agent \
79     $HOME/.cache/pip/http \
80     $HOME/.indy_client \
81     $HOME/ledger/sandbox/data \
82     $HOME/log
83
84 # The root group needs access the directories under $HOME/.indy_client and $HOME/ .
85 #   acapy_agent for the container to function in OpenShift.
86 RUN chown -R $user:root $HOME/.indy_client $HOME/.acapy_agent && \
87     chmod -R ug+rwx $HOME/log $HOME/ledger $HOME/.acapy_agent $HOME/.cache $HOME/ .
88     $HOME/.indy_client
89
90 # Create /home/indy and symlink .indy_client folder for backwards compatibility with
91 #   artifacts created on older indy-based images.
92 RUN mkdir -p /home/indy
93 RUN ln -s /home/aries/.indy_client /home/indy/.indy_client
94
95 # Install ACA-py from the wheel as $user,
96 # and ensure the permissions on the python 'site-packages' and $HOME/.local folders
97 #   are set correctly.
98 USER $user
99 COPY --from=build /src/dist/acapy_agent*.whl .
100 RUN acapy_agent_package=$(find ./ -name "acapy_agent*.whl" | head -n 1) && \
101     echo "Installing ${acapy_agent_package} ..." && \
102     pip install --no-cache-dir --find-links=. ${acapy_agent_package}${acapy_reqs} && \
103     rm acapy_agent*.whl && \
104     chmod +rx $(python -m site --user-site) $HOME/.local
105
106 ENTRYPOINT ["aca-py"]

```

Anhang 3.2.8: SSI-Agenten

Die Agent-Konfiguration in Listing A-8 erfolgt vollständig deklarativ über Docker-Compose-Service-Definitionen, die jeweils den „start“-Befehl von ACA-Py mit rollenspezifischen Parametern aufrufen. Zentrale Konfigurationsparameter umfassen „–label“ (zur Identifikation des Agents), „–inbound-transport http 0.0.0.0 <port>“ (zur Definition des DIDComm-Message-Endpoints), „–outbound-transport http“ (für ausgehende Verbindungen), „–admin 0.0.0.0 <port>“ (zur Aktivierung der Admin-API) sowie „–wallet-type askar“ (zur Spezifikation des Wallet-Backends Aries Askar).

Ein kritischer Konfigurationsparameter ist „–endpoint https://host.docker.internal:<port>“, der spezifiziert, unter welcher URL der Agent für eingehende DIDComm-Verbindungen erreichbar ist. Dieser Parameter wird in Out-of-Band-Invitations und DID-Exchange-Nachrichten eingebettet und muss auf den externen PQC-Proxy-Endpoint zeigen, nicht auf den internen Agent-Container. Beispielsweise konfiguriert der Issuer-Agent „–endpoint https://host.docker.internal:8020“, wodurch andere Agents ihre DIDComm-Nachrichten an den PQC-Proxy auf Port 8020 senden, der diese nach TLS-Terminierung an den internen Issuer-Container weiterleitet. Diese Indirektion ist essentiell, um sicherzustellen, dass alle Agent-zu-Agent-Verbindungen die Post-Quantum-gesicherte Transport-Layer-Sicherheit nutzen.

Die Wallet-Konfiguration erfolgt über die Parameter „–wallet-name <name>“, „–wallet-key <key>“ und „–auto-provision“, wobei letzterer sicherstellt, dass das Wallet automatisch beim ersten Start initialisiert wird. Jedes Wallet wird in einem dedizierten Docker-Volume persistiert („issuer-data“, „holder-data“, „verifier-data“), sodass DID-Keypairs, Credentials und Connections über Container-Neustarts hinweg erhalten bleiben.

Ein wesentlicher Aspekt der Agent-Konfiguration ist die Anbindung an die DLT-Infrastruktur über den Parameter „–genesis-url https://host.docker.internal:8000/genesis“. Diese URL referenziert den PQC-gesicherten Webserver der DLT-Infrastruktur und ermöglicht es den Agents, beim Start die Genesis-Transaktionsdatei über eine quantensichere TLS-1.3-Verbindung abzurufen. Analog dazu wird die Verbindung zur Revocation Registry über „–tails-server-base-url https://host.docker.internal:6543“ konfiguriert, wobei ebenfalls der nginx-PQC-Sidecar-Proxy als Endpoint fungiert. Die Verwendung von „host.docker.internal“ ermöglicht dabei die Adressierung des Docker-Hosts aus Container-Perspektive und abstrahiert plattformspezifische Netzwerk-Unterschiede zwischen Linux, macOS und Windows.

Zusätzlich zu den grundlegenden Konfigurations-Parametern aktivieren die Agents mehrere Auto-Response-Features, die für Entwicklungs- und Testzwecke die manuelle Interaktion reduzieren: „–auto-accept-invites“ (automatisches Akzeptieren von Connection-Invitations), „–auto-respond-credential-proposal“, „–auto-respond-credential-offer“ und „–auto-respond-credential-request“ (automatische Credential-Exchange-Antworten beim Issuer), „–auto-store-credential“ (automatisches Speichern empfangener Credentials beim Holder) sowie „–auto-verify-presentation“ (automatische Presentation-Verifikation beim Verifier). Diese Automatisierungen ermöglichen vollständig scriptgesteuerte SSI-Workflows über die Admin-API, wie sie in Jupyter-Notebook-basierten Demonstrationen implementiert sind.

Für die Post-Quantum-Absicherung der Agent-Kommunikation wurde analog zur DLT-Infrastruktur und Revocation Registry ein dedizierter nginx-PQC-Sidecar-Proxy pro Agent implementiert. Die Docker-Compose-Konfiguration definiert drei zusätzliche Services – „pqc-sidecarproxy-issuer“, „pqc-sidecarproxy-holder“ und „pqc-sidecarproxy-verifier“ –, die jeweils als Reverse Proxy vor dem zugehörigen Agent platziert werden. Jeder Agent-Container exponiert zwei interne HTTP-Ports: einen für Inbound-Transport (8020, 8030, 8040) und einen für die Admin-API (8021, 8031, 8041). Diese Ports sind ausschließlich im agent-spezifischen internen Docker-Netzwerk („hope-issuer“, „hope-holder“, „hope-verifier“) zugänglich.

Die Sidecar-Proxies terminieren alle eingehenden TLS-1.3-Verbindungen und leiten die Anfragen nach erfolgreicher quantensicherer Authentifizierung und Schlüsselvereinbarung als unverschlüsseltes HTTP an die internen Agent-Ports weiter. Jeder Proxy ist mit zwei Docker-Netzwerken verbunden, dem agent-spezifischen internen Netzwerk für Backend-Kommunikation sowie dem gemeinsamen externen „von_sidecarproxy“-Netzwerk für Client-Zugriffe. Diese Dual-Network-Architektur erzwingt, dass sämtliche externe Kommunikation – einschließlich DIDComm-Nachrichten zwischen Agents, Admin-API-Zugriffe durch Controller-Anwendungen sowie Ledger- und Tails-Server-Requests – über quantensichere TLS-Verbindungen erfolgt.

Die Netzwerk-Architektur folgt einem strikten Isolation-Prinzip. Jeder Agent residiert in einem dedizierten internen Docker-Netzwerk, das ausschließlich den Agent-Container und den zugehörigen PQC-Proxy umfasst. Die Agents können untereinander nicht direkt kommunizieren, sondern ausschließlich über das externe „von_sidecarproxy“-Netzwerk, das die PQC-Sidecar-Proxies verbindet. Diese Segmentierung erzeugt eine „Defense in Depth“-Architektur nach Alsaqour et al. (2021, S. 242–243), bei der selbst bei einer Kompromittierung eines Agent-Containers der Zugriff auf andere Agents durch Netzwerk-Isolation verhindert wird.

Die Health-Check-Konfiguration der Agent-Container überwacht die Verfügbarkeit der Admin-API mittels periodischer HTTP-Requests an „`http://localhost:<admin-port>/status/ready`“. Zusätzlich definieren die Service-Dependencies („depends_on“) eine Startup-Reihenfolge, die sicherstellt, dass die PQC-Sidecar-Proxies vor den Agents starten und dass die Infrastruktur-Services (von-network, Tails-Server) vollständig initialisiert sind, bevor die Agents ihre Genesis-Datei abrufen. Diese Orchestrierung eliminiert Race-Conditions während des Deployment-Prozesses und gewährleistet eine deterministische Startup-Sequenz.

Listing A-8

docker-compose.yml: SSI-Agenten

```

1
2 version: '3.8'
3
4 services:
5   issuer:
6     build:
7       context: ..
8       dockerfile: hopE/Dockerfile.acapy-base
9     image: acapy-base
10    container_name: issuer-agent
11    environment:
12      - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
13      - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
14      - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
15      - PUBLIC_TAILS_URL=https://host.docker.internal:6543
16      - TAILS_FILE_COUNT=100
17    networks:
18      - hope-issuer
19    volumes:
20      - issuer-data:/home/aries/.acapy_agent
21    extra_hosts:
22      - "host.docker.internal:host-gateway"
23    command: >
24      start
25      --label "Issuer Agent"
26      --inbound-transport http 0.0.0.0 8020
27      --outbound-transport http
28      --endpoint https://host.docker.internal:8020
29      --admin 0.0.0.0 8021
30      --admin-insecure-mode
31      --auto-provision
32      --wallet-type askar
33      --wallet-name issuer_wallet
34      --wallet-key issuer_wallet_key_000000000000
35      --genesis-url https://host.docker.internal:8000/genesis
36      --log-level info
37      --auto-accept-invites
38      --auto-accept-requests
39      --auto-ping-connection
40      --auto-respond-credential-proposal
41      --auto-respond-credential-offer

```

```

42      --auto-respond-credential-request
43      --auto-verify-presentation
44      --public-invites
45      --preserve-exchange-records
46      --tails-server-base-url https://host.docker.internal:6543
47      --notify-revocation
48  healthcheck:
49    test: ["CMD", "curl", "-f", "http://localhost:8021/status/ready"]
50    interval: 30s
51    timeout: 10s
52    retries: 5
53    start_period: 30s
54
55  holder:
56    build:
57      context: ..
58      dockerfile: hopE/Dockerfile.acapy-base
59    image: acapy-base
60    container_name: holder-agent
61    environment:
62      - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
63      - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
64      - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
65      - PUBLIC_TAILS_URL=https://host.docker.internal:6543
66  networks:
67    - hope-holder
68  volumes:
69    - holder-data:/home/aries/.acapy_agent
70  extra_hosts:
71    - "host.docker.internal:host-gateway"
72  command: >
73    start
74    --label "Holder Agent"
75    --inbound-transport http 0.0.0.0 8030
76    --outbound-transport http
77    --endpoint https://host.docker.internal:8030
78    --admin 0.0.0.0 8031
79    --admin-insecure-mode
80    --auto-provision
81    --wallet-type askar
82    --wallet-name holder_wallet
83    --wallet-key holder_wallet_key_000000000000
84    --genesis-url https://host.docker.internal:8000/genesis
85    --log-level info
86    --auto-accept-invites
87    --auto-accept-requests
88    --auto-ping-connection
89    --auto-respond-credential-offer
90    --auto-store-credential
91    --public-invites
92    --tails-server-base-url https://host.docker.internal:6543
93    --preserve-exchange-records
94  healthcheck:
95    test: ["CMD", "curl", "-f", "http://localhost:8031/status/ready"]
96    interval: 30s
97    timeout: 10s
98    retries: 5

```

```

99      start_period: 30s
100
101 verifier:
102   build:
103     context: ..
104     dockerfile: hopE/Dockerfile.acapy-base
105   image: acapy-base
106   container_name: verifier-agent
107   environment:
108     - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
109     - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
110     - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
111     - PUBLIC_TAILS_URL=https://host.docker.internal:6543
112 networks:
113   - hope-verifier
114 volumes:
115   - verifier-data:/home/aries/.acapy_agent
116 extra_hosts:
117   - "host.docker.internal:host-gateway"
118 command: >
119   start
120   --label "Verifier Agent"
121   --inbound-transport http 0.0.0.0 8040
122   --outbound-transport http
123   --endpoint https://host.docker.internal:8040
124   --admin 0.0.0.0 8041
125   --admin-insecure-mode
126   --auto-provision
127   --wallet-type askar
128   --wallet-name verifier_wallet
129   --wallet-key verifier_wallet_key_000000000000
130   --genesis-url https://host.docker.internal:8000/genesis
131   --log-level info
132   --auto-accept-invites
133   --auto-accept-requests
134   --auto-ping-connection
135   --auto-verify-presentation
136   --public-invites
137   --tails-server-base-url https://host.docker.internal:6543
138   --preserve-exchange-records
139 healthcheck:
140   test: ["CMD", "curl", "-f", "http://localhost:8041/status/ready"]
141   interval: 30s
142   timeout: 10s
143   retries: 5
144   start_period: 30s
145
146 pqc-sidecarproxy-issuer:
147   build:
148     context: ./pqc_sidecarproxy_nginx
149     dockerfile: Dockerfile
150     args:
151       OPENSSL_TAG: openssl-3.5.4
152       LIBOQS_TAG: 0.13.0
153       OQSPROVIDER_TAG: 0.9.0
154       NGINX_VERSION: 1.28.0
155       SIG_ALG: mldsa65

```

```

156      DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
157      container_name: pqc-sidecarproxy-issuer
158      environment:
159          # OpenSSL Configuration
160          - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
161          # Post-Quantum Key Exchange Groups
162          - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
163      networks:
164          - von_sidecarproxy
165          - hope-issuer
166      ports:
167          - "8020:8020" # Issuer Inbound Transport HTTPS (ML-KEM-768)
168          - "8021:8021" # Issuer Admin API HTTPS (ML-KEM-768)
169      volumes:
170          # Custom nginx configuration for reverse proxy
171          - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_issuer.conf:/opt/nginx/nginx-conf/
172              nginx.conf:ro
173          # Custom ML-DSA-65 certificates
174          - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
175          # Logs
176          - nginx-logs:/opt/nginx/logs
177      depends_on:
178          - issuer
179          - holder
180          - verifier
181      restart: unless-stopped
182      healthcheck:
183          test: ["CMD", "curl", "-k", "-f", "https://localhost:8021/health"]
184          interval: 30s
185          timeout: 10s
186          retries: 5
187          start_period: 10s
188
189      pqc-sidecarproxy-holder:
190          build:
191              context: ./pqc_sidecarproxy_nginx
192              dockerfile: Dockerfile
193              args:
194                  OPENSSL_TAG: openssl-3.5.4
195                  LIBOQS_TAG: 0.13.0
196                  OQSPROVIDER_TAG: 0.9.0
197                  NGINX_VERSION: 1.28.0
198                  SIG_ALG: mldsa65
199                  DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
200          container_name: pqc-sidecarproxy-holder
201          environment:
202              # OpenSSL Configuration
203              - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
204              # Post-Quantum Key Exchange Groups
205              - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
206          networks:
207              - von_sidecarproxy
208              - hope-holder
209          ports:
210              - "8030:8030" # Holder Inbound Transport HTTPS (ML-KEM-768)
211              - "8031:8031" # Holder Admin API HTTPS (ML-KEM-768)
212          volumes:

```

```

212      # Custom nginx configuration for reverse proxy
213      - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_holder.conf:/opt/nginx/nginx-conf/
214          nginx.conf:ro
215      # Custom ML-DSA-65 certificates
216      - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
217      # Logs
218      - nginx-logs:/opt/nginx/logs
219 depends_on:
220     - issuer
221     - holder
222     - verifier
223 restart: unless-stopped
224 healthcheck:
225     test: ["CMD", "curl", "-k", "-f", "https://localhost:8031/health"]
226     interval: 30s
227     timeout: 10s
228     retries: 5
229     start_period: 10s
230
231 pqc-sidecarproxy-verifier:
232     build:
233         context: ./pqc_sidecarproxy_nginx
234         dockerfile: Dockerfile
235         args:
236             OPENSSL_TAG: openssl-3.5.4
237             LIBOQS_TAG: 0.13.0
238             OQSPROVIDER_TAG: 0.9.0
239             NGINX_VERSION: 1.28.0
240             SIG_ALG: mldsa65
241             DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
242         container_name: pqc-sidecarproxy-verifier
243         environment:
244             # OpenSSL Configuration
245             - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
246             # Post-Quantum Key Exchange Groups
247             - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
248     networks:
249         - von_sidecarproxy
250         - hope-verifier
251     ports:
252         - "8040:8040"  # Verifier Inbound Transport HTTPS (ML-KEM-768)
253         - "8041:8041"  # Verifier Admin API HTTPS (ML-KEM-768)
254     volumes:
255         # Custom nginx configuration for reverse proxy
256         - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_verifier.conf:/opt/nginx/nginx-conf/
257             nginx.conf:ro
258         # Custom ML-DSA-65 certificates
259         - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
260         # Logs
261         - nginx-logs:/opt/nginx/logs
262 depends_on:
263     - issuer
264     - holder
265     - verifier
266 restart: unless-stopped
267 healthcheck:
268     test: ["CMD", "curl", "-k", "-f", "https://localhost:8041/health"]

```

```

267     interval: 30s
268     timeout: 10s
269     retries: 5
270     start_period: 10s
271
272 networks:
273   hope-issuer:
274   hope-holder:
275   hope-verifier:
276   von_sidecarproxy:
277     external: true
278
279 volumes:
280   issuer-data:
281   holder-data:
282   verifier-data:
283   nginx-logs:

```

Anhang 3.2.9: Docker Orchestrierung der Gesamtarchitektur

Listing A-9

Docker Compose Start der Gesamtarchitektur

```

1 (.venv) ferris@blockchain-ssi-pqc:~/github/MSc-blockchain-ssi-pqc$ ./von-network/
      manage start && ./indy-tails-server/docker/manage start && docker compose -f ./
      hopE/docker-compose.yml up -d
2 [+]
3 Network von_von                         Created    0.0s
4 Network von_sidecarproxy                  Created    0.0s
5 Volume "von_webserver-ledger"             Created    0.0s
6 Volume "von_node1-data"                   Created    0.0s
7 Volume "von_node3-data"                   Created    0.0s
8 Volume "von_node2-data"                   Created    0.0s
9 Volume "von_node4-data"                   Created    0.0s
10 Volume "von_nginx-logs"                  Created    0.0s
11 Volume "von_webserver-cli"               Created    0.0s
12 Container von-node4                     Started    0.6s
13 Container von-webserver                 Started    0.4s
14 Container von-node3                     Started    0.4s
15 Container von-node2                     Started    0.5s
16 Container von-node1                     Started    0.5s
17 Container von-pqc-sidecarproxy-webserver Started    0.6s
18 Want to see the scrolling container logs? Run "./manage logs"
19 [+]
20 Network docker_tails-server              Created    0.0s
21 Volume "docker_nginx-logs"               Created    0.0s
22 Container docker-tails-server-1         Started    0.3s
23 Container pqc-sidecarproxy-tails-server Started    0.5s
24 Run './manage logs' for logs
25 WARN[0000] /home/ferris/github/MSc-blockchain-ssi-pqc/hopE/docker-compose.yml: the
      attribute 'version' is obsolete, it will be ignored, please remove it to avoid
      potential confusion
26 [+]

```

27	Network hope_hope-issuer	Created	0.0s
28	Network hope_hope-holder	Created	0.0s
29	Network hope_hope-verifier	Created	0.0s
30	Volume "hope_issuer-data"	Created	0.0s
31	Volume "hope_holder-data"	Created	0.0s
32	Volume "hope_verifier-data"	Created	0.0s
33	Volume "hope_nginx-logs"	Created	0.0s
34	Container issuer-agent	Started	0.6s
35	Container holder-agent	Started	0.5s
36	Container verifier-agent	Started	0.5s
37	Container pqc-sidecarproxy-holder	Started	1.3s
38	Container pqc-sidecarproxy-issuer	Started	1.0s
39	Container pqc-sidecarproxy-verifier	Started	1.2s

Listing A-10

von-network manage Skript

```

1  #!/bin/bash
2  export MSYS_NO_PATHCONV=1
3
4  # Default Options
5  export TA_RATIFICATION_TIME_OPS='+%s --date='
6  export STAT_OPS='-c "%a"'
7
8  # MAC OS Options
9  if [[ $OSTYPE == 'darwin'* ]]; then
10    # Set default platform to linux/amd64 when running on Arm based MAC since there are
11    # no arm based images available currently.
12    architecture=$(uname -m)
13    if [[ "${architecture}" == 'arm'* ]] || [[ "${architecture}" == 'aarch'* ]]; then
14      export DOCKER_DEFAULT_PLATFORM=linux/amd64
15    fi
16
17    # Set the date and stat options appropriately for MAC OS.
18    export TA_RATIFICATION_TIME_OPS='--jf "%Y-%m-%dT%H:%M:%S%Z" +%s'
19    export STAT_OPS='--f "%A"'
20  fi
21
22  # getDockerHost; for details refer to https://github.com/bcgov/DITP-DevOps/tree/main/
23  # code/snippets#getdockerhost
24  . /dev/stdin <<<$(cat <(curl -s --raw https://raw.githubusercontent.com/bcgov/DITP-
25  DevOps/main/code/snippets/getDockerHost))"
26  export DOCKERHOST=$(getDockerHost)
27
28  SCRIPT_HOME="$( cd "$( dirname "$0" )" && pwd )"
29  export COMPOSE_PROJECT_NAME="${COMPOSE_PROJECT_NAME:-von}"
30
31  export TMP_FOLDER='./tmp'
32  export CLI_SCRIPTS_FOLDER='./cli-scripts'
33  export DEFAULT_CLI_SCRIPT_DIR="${CLI_SCRIPTS_FOLDER}"
34
35  export LEDGER_TIMEOUT="${LEDGER_TIMEOUT:-60}"
36  export LEDGER_URL_CONFIG="${LEDGER_URL_CONFIG}"
37
38  export ROOT_BACKUP_DIR=backup
39  export SHELL_CMD='bash'
```



```

87
88 stop - Stops the services. This is a non-destructive process. The volumes and
89 containers
90     are not deleted so they will be reused the next time you run start.
91
92 rebuild - Rebuild the docker images.
93
94 dockerhost - Print the ip address of the Docker Host Adapter as it is seen by
95     containers running in docker.
96
97 generateSecrets - Generate a random set of secrets using openssl; a Seed and a Key.
98
99 generateDid - Generates a DID and Verkey from a Seed.
100    $0 generateDid [seed]
101        - Optional [seed]; if one is not provided a random one will be generated
102            using openssl.
103
104 generateGenesisFiles - Generates pool and domain genesis files from data input via
105     csv files.
106    $0 generategenesisfiles <trustees_csv_file> <stewards_csv_file>
107
108 This is a convenience command wrapped around the Steward Tools script for
109     generating genesis files found here;
110     https://github.com/sovrin-foundation/steward-tools/tree/master/create\_genesis
111
112 The script is downloaded and hosted in a running container which has the
113     required packages installed.
114
115 Examples of the csv files can be downloaded from here;
116     https://docs.google.com/spreadsheets/d/1LDduIeZp7pansd9deXeVSqGgdf0VdAHNMc7xYli3QAY/edit#gid=0
117 Download each sheet separately in csv format and fill them out with the data
118     specific to your network.
119
120 The input csv files must be placed into the ./tmp/ folder.
121 The resulting output 'pool_transactions_genesis' and '
122         domain_transactions_genesis' files will be placed
123     in the ./tmp/ folder.
124
125 Example:
126     $0 generategenesisfiles "./tmp/CANDy Beta Genesis File Node info - Trustees.
127         csv" "./tmp/CANDy Beta Genesis File Node info - Stewards.csv"
128
129 apply-taa - Registers an AML and TAA on a given network.
130     This is a convenient wrapper around the cli-scripts/apply-taa batch script,
131     which simplifies and automates the process of registering an AML and TAA
132
133     .
134
135     The following parameters are simply the standard inputs for the indy-cli
136         batch script.
137     Refer to the cli-scripts/apply-taa documentation for details;
138         - walletName          - Required
139         - storageType         - Optional
140         - storageConfig       - Optional
141         - storageCredentials - Optional
142         - poolName            - Required
143         - useDid              - Required

```

```

132
133     The following parameters are required:
134     amlUrl
135         - The URL to the raw content of the AML. Provided the link contains
136             the AML expected content in json format
137             (contained in {})s) it will be parsed from the link's text.
138     amlVersion
139         - The version of the AML.
140     taaUrl
141         - The URL to the raw content of the TAA in markdown format.
142     taaVersion
143         - The version of the TAA
144     taaRatificationTime
145         - The date and time of TAA ratification by network government.
146         - On Linux and Windows (Git Bash):
147             - The date format is "%Y-%m-%dT%H:%M:%S%Z", where '%Z' is a numeric
148                 time zone offset, or an alphabetic time zone abbreviation.
149             - Examples:
150                 - "2022-08-09T11:05:00-0700"
151                 - "2022-08-09T11:05:00PDT"
152         - On Mac:
153             - The date format is "%Y-%m-%dT%H:%M:%S%Z", where '%Z' is an
154                 alphabetic time zone abbreviation.
155             - Example:
156                 - "2022-08-09T11:05:00PDT"
157
158     Example:
159     $0 apply-taa \
160         walletName=local_net_trustee_wallet \
161         poolName=local_net \
162         useDid=V4SGRU86Z58d6TV7PBUE6f \
163         amlUrl='https://raw.githubusercontent.com/bcgov/bc-vcpedia/(Layer
164             -1)-CANDy-Acceptance-Mechanism-List-(AML).md' \
165         amlVersion=0.1 \
166         taaUrl='https://raw.githubusercontent.com/bcgov/bc-vcpedia/(Layer
167             -1)-CANDy-Transaction-Author-Agreement.md' \
168         taaVersion=0.1 \
169         taaRatificationTime="2022-08-09T11:05:00PDT"
170
171     truncateLogs - Truncate the container logs.
172
173     indy-cli - Run Indy-Cli commands in a Indy-Cli container environment.
174
175     $0 indy-cli -h
176         - Display specific help documentation.
177
178     cli - Run a command in an Indy-Cli container.
179
180     $0 cli -h
181         - Display specific help documentation.
182
183     backup [description] - Backup the current von-network environment.
184         Creates a set of tar.gz archives of each of the environment's volumes.
185         Backup sets are stored in a ./backup/date/time folder structure.
186         Examples:
187             $0 backup
188             $0 backup "The description of my environment's current state."

```

```

184           - Make a backup and include the description in a ReadMe.txt file.
185
186 restore [filter] - Restore a given backup set. Defaults to restoring the most
187   recent backup.
188   Examples:
189   $0 restore
190     - Restore the most recent backup.
191   $0 restore 13-15-37
192     - Restore the backup from 13-15-37 today.
193   $0 restore von_client-data_2021-08-23_08-21-08.tar
194     - Restore the backup set containing the von_client-data_2021-08-23_08
195       -21-08.tar archive.
196   $0 restore 2021-08-23
197     - Restore the most recent backup set from 2021-08-23.

198 restoreArchive <archive> <volume> [tarOptions]- Restore a tar.gz archive to a named
199   volume.
200   Useful for restoring an archive for inspection and debugging purposes.
201   Examples:
202   $0 restoreArchive ./backup/Node3-ledger-backup.tar.gz node3-bcovrin-tes
203   $0 restoreArchive ./backup/Node3-ledger-backup.tar.gz node3-bcovrin-test --
204     strip=1
205     - Restore the archive to the named volume, stripping the first level
206       directory from the archive.
207     - Useful in the scenario where the archive contains additional directory
208       levels that aren't needed in the restored copy.

209 debugVolume <volume> [volumeMountFolder] - Mount a named volume into a 'debug'
210   instance of the 'von-network-base' image with an interactive shell.
211   Provides a containerized environment to perform analysis on the ledger
212     databases and files.
213   Starting with 'von-image:node-1.12-4' the base image for von-network
214     contains the RocksDB sst_dump tool that can be used to verify
215     and inspect the RocksDB database files; '*.sst' files.
216   For example the command 'find /debug_volume/ -name "*.sst" | xargs -I {}
217     sst_dump --file={} --command=verify' can be used to do a quick
218     verification on all the database files once the container starts.
219   Usage information for sst_dump can be found here; https://github.com/
220     facebook/rocksdb/wiki/Administration-and-Data-Access-Tool
221   Examples:
222   $0 debugVolume node3-bcovrin-test
223   $0 debugVolume node1-bcovrin-test /home/indy/ledger
224     - Mount the named volume to '/home/indy/ledger'

225 EOF
226 exit 1
227 }
228 }

229 indyCliUsage () {
230   cat <<-EOF
231
232   Usage:
233   $0 [options] indy-cli [-h] [command] [parameters]
234
235   Run Indy-Cli commands in a Indy-Cli container environment.
236   - Refer to the cli-scripts directory for available scripts and their parameters.
237   - Refer to './docs/Writing Transactions to a Ledger for an Un-privileged Author.
238     md' for

```

```

229         additional examples.
230
231     Options:
232     -v <FullyQualifiedPathToScripts/>
233         - Mount a script volume to the container. By default the 'cli-scripts'
234             directory is mounted to the container.
235
236     Examples:
237
238     $0 indy-cli
239         - Start an interactive indy-cli session in your Indy-Cli Container.
240
241     $0 indy-cli --help
242         - Get usage information for the indy-cli.
243 EOF
244 exit 1
245 }
246
247 cliUsage () {
248     cat <<-EOF
249
250     Usage:
251     $0 [options] cli [-h] [command]
252
253     Run a command in an Indy-Cli container.
254
255     Options:
256     -v <FullyQualifiedPathToScripts/>
257         - Mount a script volume to the container. By default the 'cli-scripts'
258             directory is mounted to the container.
259
260     Examples:
261
262     $0 cli reset
263         - Reset your Indy-CLI container's environment
264
265     $0 cli init-pool localpool http://192.168.65.3:9000/genesis
266     $0 cli init-pool MainNet https://raw.githubusercontent.com/sovrin-foundation/
267         sovrin/stable/sovrin/pool_transactions_genesis
268         - Initialize the pool for your Indy-CLI container's environment.
269 EOF
270 exit 1
271 }
272
273 # -----
274 # Initialization:
275 # -----
276 while getopts v:h FLAG; do
277     case $FLAG in
278         v ) VOLUMES=$OPTARG ;;
279         h ) usage ;;
280         \? ) #unrecognized option - show help
281             echo -e \\n"Invalid script option: -${OPTARG}"\\n
282             usage
283             ;;
284     esac
285 done

```

```

283 shift $((OPTIND-1))
284
285 # -----
286 # Functions:
287 # -----
288 function toLower() {
289   echo ${echo ${@} | tr '[:upper:]' '[lower:]'}
290 }
291
292 function initDockerBuildArgs() {
293   dockerBuildArgs=""
294
295   # HTTP proxy, prefer lower case
296   if [[ "${http_proxy}" ]]; then
297     dockerBuildArgs+=" ${dockerBuildArgs} --build-arg http_proxy=${http_proxy}"
298   else
299     if [[ "${HTTP_PROXY}" ]]; then
300       dockerBuildArgs+=" ${dockerBuildArgs} --build-arg http_proxy=${HTTP_PROXY}"
301     fi
302   fi
303
304   # HTTPS proxy, prefer lower case
305   if [[ "${https_proxy}" ]]; then
306     dockerBuildArgs+=" ${dockerBuildArgs} --build-arg https_proxy=${https_proxy}"
307   else
308     if [[ "${HTTPS_PROXY}" ]]; then
309       dockerBuildArgs+=" ${dockerBuildArgs} --build-arg https_proxy=${HTTPS_PROXY}"
310     fi
311   fi
312
313   echo ${dockerBuildArgs}
314 }
315
316 function initEnv() {
317
318   if [ -f .env ]; then
319     while read line; do
320       if [[ ! "$line" =~ ^\# ]] && [[ "$line" =~ .*= ]]; then
321         export ${line//[$'\r\n']}
322       fi
323     done <.env
324   fi
325
326   for arg in "$@"; do
327     # Remove recognized arguments from the list after processing.
328     shift
329     case "$arg" in
330       *=*)
331       export "${arg}"
332       ;;
333       --logs)
334         TAIL_LOGS=1
335         ;;
336       --wait)
337         WAIT_FOR_LEDGER=1
338         ;;
339       --taa-sample)

```

```

340     USE_SAMPLE_TAA=1
341     ;;
342   *)
343     # If not recognized, save it for later processing ...
344     set -- "$@" "$arg"
345     ;;
346   esac
347 done
348
349 IP=""
350 IPS=""
351 if [ ! -z $(echo ${1} | grep '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' ) ]; then
352   if [[ $1 == *,"* ]]; then
353     IPS="$1"
354   else
355     IP="$1"
356   fi
357 fi
358 export IP="$IP" IPS="$IPS"
359
360 export LEDGER_SEED=${LEDGER_SEED}
361
362 export LOG_LEVEL=${LOG_LEVEL:-info}
363 export RUST_LOG=${RUST_LOG:-warning}
364 }
365
366 function runCliCommand() {
367
368   unset displayCliUsage
369   for arg in "$@"; do
370     # Remove recognized arguments from the list after processing.
371     shift
372     case "$arg" in
373       -h)
374         displayCliUsage=1
375         ;;
376     *)
377       # If not recognized, save it for later processing ...
378       set -- "$@" "$arg"
379       ;;
380     esac
381   done
382
383   initEnv "$@"
384   cliCmd="${1}"
385   shift || cliCmd=""
386
387   if [ ! -z "${displayCliUsage}" ] && [[ "${cliCmd}" == "indy-cli" ]]; then
388     indyCliUsage
389   elif [ ! -z "${displayCliUsage}" ] && [[ -z "${cliCmd}" ]]; then
390     cliUsage
391   fi
392
393   cmd="${dockerCompose} \
394     run "
395

```

```

396  if [ -z "${VOLUMES}" ] && [ -d "${DEFAULT_CLI_SCRIPT_DIR}" ] ; then
397      VOLUMES=$(realpath ${DEFAULT_CLI_SCRIPT_DIR})
398  fi
399
400  if [ ! -z "${VOLUMES}" ]; then
401      shopt -s extglob
402      paths=$(echo "${VOLUMES}" | sed -n 1'p' | tr ',' '\n')
403      for path in ${paths}; do
404          path=$(path%%+(/))
405          mountPoint=${path##*/}
406          if [[ "$OSTYPE" == "msys" ]]; then
407              # When running on Windows, you need to prefix the path with an extra '/'
408              path="/${path}"
409          fi
410          cmd+=" -v '${path}:/home/indy/${mountPoint}:Z'"
411      done
412  fi
413
414  # Need to escape quotes and commas so they don't get removed along the way ...
415  escapedArgs=$(echo $@ | sed "s'~\\'\\'g" | sed 's~"~\\\"~g' | sed 's~(~\\)(~g' | sed
416      's~)~\\)~g')
417
418  # Quote the escaped args so docker compose does not try to perform any processing on
419  # them ...
420  # Separate the command and the args so they don't get treated as one argument by the
421  # scripts in the container ...
422  cmd+="
423      --rm client \
424      ./scripts/manage ${cliCmd} \"${escapedArgs}\""
425
426  eval ${cmd}
427 }
428
429 function logs() {
430     (
431         local OPTIND
432         local unset _force
433         local unset no_tail
434         while getopts ":f:-:" FLAG; do
435             case $FLAG in
436                 f ) local _force=1 ;;
437                 - )
438                     case ${OPTARG} in
439                         "no-tail") no_tail=1
440                         ;;
441                     esac
442             esac
443         done
444         shift $((OPTIND-1))
445
446         log_args=()
447         (( no_tail != 1 )) && log_args+=(-f)
448         if [ ! -z "${TAIL_LOGS}" ] || [ ! -z "${_force}" ]; then
449             ${dockerCompose} \
450             logs \
451             "${log_args[@]}" "$@"
452         fi

```

```

450    )
451 }
452
453 pingLedger() {
454     ledger_url=${1}
455
456     # ping ledger web browser for genesis txns
457     local rtnCd=$(curl -s --write-out '%{http_code}' --output /dev/null ${ledger_url}/
458         genesis)
459     if (( ${rtnCd} == 200 )); then
460         return 0
461     else
462         return 1
463     fi
464 }
465
466 function wait_for_ledger() {
467 (
468     # if flag is set, wait for ledger to activate before continuing
469     local rtnCd=0
470     if [ ! -z "${WAIT_FOR_LEDGER}" ]; then
471         # Wait for ledger server to start ...
472         local startTime=${SECONDS}
473         # use global LEDGER_URL
474         local LEDGER_URL="${LEDGER_URL_CONFIG:-http://localhost:9000}"
475         printf "waiting for ledger to start"
476         while ! pingLedger "$LEDGER_URL"; do
477             printf "."
478             local duration=$((SECONDS - startTime))
479             if (( ${duration} >= ${LEDGER_TIMEOUT} )); then
480                 echoRed "\nThe Indy Ledger failed to start within ${duration} seconds.\n"
481                 rtnCd=1
482                 break
483             fi
484             sleep 1
485         done
486     fi
487     return ${rtnCd}
488 )
489
490 function install_taa() {
491 (
492     # if flag is set, copy the sample config/sample_aml.json and config/sample_taa.
493     # json
494     # to config/aml.json and config/taa.json. Also create a marker file so that we
495     # clean up on shutdown and still support backward compatibility where people
496     # previously
497     # their own custom versions and don't want them removed.
498     local rtnCd=0
499     if [ ! -z "${USE_SAMPLE_TAA}" ]; then
500         rtnCd=$(cp -f ./config/sample_aml.json ./config/aml.json) && $(cp -f ./config/
501             sample_taa.json ./config/taa.json)
502         touch ./config/.samples.used
503     fi
504     return ${rtnCd}
505 )

```

```

503 }
504
505 function remove_taa() {
506 (
507 # if the marker exists indicating we created the aml and taa files, make sure we
508 # remove them to clean up.
509 if [ -f "./config/.samples.used" ]; then
510 rm -f ./config/aml.json ./config/taa.json ./config/.samples.used
511 fi
512 )
513
514 function generateKey() {
515 (
516 _length=${1:-48}
517 # Format can be `--base64` or `--hex`
518 _format=${2:--base64}
519 echo $(openssl rand ${_format} ${_length})
520 )
521 }
522
523 function generateSeed() {
524 (
525 _prefix=${1}
526 _seed=$(echo "${_prefix}$(generateKey 32)" | fold -w 32 | head -n 1 )
527 _seed=$(echo -n "${_seed}")
528 echo ${_seed}
529 )
530 }
531
532 function generateSecrets() {
533 echo
534 echo "Seed: $(generateSeed)"
535 echo "Key: $(generateKey)"
536 echo
537 }
538
539 function generateDid() {
540 seed=${1}
541 if [ -z ${seed} ]; then
542 seed=$(generateSeed)
543 fi
544 runCliCommand python cli-scripts/generate_did.py --seed ${seed}
545 }
546
547 function generateGenesisFiles() {
548 trustee_csv="${1}"
549 steward_csv="${2}"
550 genesis_from_files_filename="genesis_from_files.py"
551 genesis_from_files_url="https://raw.githubusercontent.com/sovrin-foundation/steward-
552 tools/master/create_genesis/genesis_from_files.py"
553 if [ -z "${trustee_csv}" ] || [ -z "${steward_csv}" ]; then
554 echoYellow "You must supply both the trustee and steward csv files."
555 exit 1
556 fi
557 }
```

```

558     if [[ "${trustee_csv}" != ${TMP_FOLDER}/* ]]; then
559         trustee_csv="${TMP_FOLDER}/${trustee_csv}"
560     fi
561
562     if [ ! -f "${trustee_csv}" ]; then
563         echoYellow "${trustee_csv} not found, please make sure you placed ${trustee_csv}
564             in the ${TMP_FOLDER} folder."
565         exit 1
566     fi
567
568     if [[ "${steward_csv}" != ${TMP_FOLDER}/* ]]; then
569         steward_csv="${TMP_FOLDER}/${steward_csv}"
570     fi
571
572     if [ ! -f "${steward_csv}" ]; then
573         echoYellow "${steward_csv} not found, please make sure you placed ${steward_csv}
574             in the ${TMP_FOLDER} folder."
575         exit 1
576     fi
577
578     if [ ! -f "${CLI_SCRIPTS_FOLDER}/${genesis_from_files_filename}" ]; then
579         echo "Downloading the latest version of ${genesis_from_files_filename} from ${
580             genesis_from_files_url} ..."
581         curl -s -L -o ${CLI_SCRIPTS_FOLDER}/${genesis_from_files_filename} ${
582             genesis_from_files_url}
583         chmod +x ${CLI_SCRIPTS_FOLDER}/${genesis_from_files_filename}
584     fi
585
586     # Escape spaces in path ...
587     trustee_csv_esc=$(echo ${trustee_csv##*/} | sed 's/ /\ \ /g')
588     steward_csv_esc=$(echo ${steward_csv##*/} | sed 's/ /\ \ /g')
589     runCliCommand cli-scripts/${genesis_from_files_filename} --pool /tmp/
590         pool_transactions --domain /tmp/domain_transactions --trustees /tmp/${{
591             trustee_csv_esc} --stewards /tmp/${steward_csv_esc}
592 }
593
594 function apply-taa() {
595     # Parse Args
596     for arg in "$@"; do
597         # Remove recognized arguments from the list after processing.
598         shift
599         case "${arg}" in
600             *=*)
601                 if [[ "${arg}" == *"amlUrl"* ]]; then
602                     amlUrl=${arg#*=}
603                     continue
604                 fi
605
606                 if [[ "${arg}" == *"taaUrl"* ]]; then
607                     taaUrl=${arg#*=}
608                     continue
609                 fi
610
611                 if [[ "${arg}" == *"taaRatificationTime"* ]]; then
612                     taaRatificationTime=${arg#*=}
613                     taaRatificationTimestamp=$(date ${TA_RATIFICATION_TIME_OPS}"${{
614                         taaRatificationTime})"

```

```

608         continue
609     fi
610
611     batchFileArgs+=" ${arg}"
612     ;;
613   esac
614 done
615
616 # Create temp files for the AML and ATT
617 amlPath="${TMP_FOLDER}/aml-${date}+%s.txt"
618 taaPath="${TMP_FOLDER}/att-${date}+%s.txt"
619
620 # Download the AML and ATT files
621 curl -s -o ${amlPath} "${amlUrl}"
622 curl -s -o ${taaPath} "${taaUrl}"
623
624 # Parse the AML content from the file; the bit inside the '{}'
625 amlContent=$(sed -n '/{/,/}/ { /{/{/!p;};}' ${amlPath} | sed 's/^ * */g' )
626 echo ${amlContent} > ${amlPath}
627
628 # Generate the required apply-taa batch script parameters
629 batchFileArgs+=" amlFile=/tmp/${amlPath##*/}"
630 batchFileArgs+=" amlContext=${amlUrl}"
631 batchFileArgs+=" taaFile=/tmp/${taaPath##*/}"
632 batchFileArgs+=" taaRatificationTimestamp=${taaRatificationTimestamp}"
633
634 # Run the apply-taa batch script
635 runCliCommand indy-cli --config /tmp/cliconfig.json apply-taa \
636                 ${batchFileArgs}
637
638 # Remove the temp files
639 rm "${amlPath}" "${taaPath}"
640 }
641
642 function backup() {
643 (
644 _msg=$@
645 volumes=$(${dockerCompose} config --volumes)
646 timeStamp='date +\%Y-\%m-\%d_%H-\%M-\%S'
647 datePart=${timeStamp%_*}
648 timePart=${timeStamp#_*}
649 backupDir=${ROOT_BACKUP_DIR}/${datePart}/${timePart}
650 backupVolumeMount=$(getVolumeMount ./${ROOT_BACKUP_DIR})/
651 mkdir -p ./${backupDir}
652 chmod -R ug+rw ./${backupDir}
653 if [ ! -z "${_msg}" ]; then
654   echo "${_msg}" > ./${backupDir}/ReadMe.txt
655 fi
656
657 for volume in ${volumes}; do
658   volume=$(echo ${volume} | sed 's~\r$~~')
659   sourceVolume=${COMPOSE_PROJECT_NAME}_${volume}
660   archiveName=${sourceVolume}_${timeStamp}.tar.gz
661   archivePath="/${backupDir}/${archiveName}"
662
663 echoYellow \\n"Backing up ${sourceVolume} to ${archivePath} ..."
664 docker run \

```

```

665      --rm \
666      --name von-network-backup \
667      -v ${backupVolumeMount}:/${ROOT_BACKUP_DIR} \
668      -v ${sourceVolume}:/source_volume von-network-base \
669      tar -czvf ${archivePath} -C /source_volume/ .
670      done
671    )
672  }
673
674 function restore() {
675  (
676    _fileName=${1}
677    archivePath=$(findBackup ${1})
678    archiveDirectory=${archivePath%/*}
679    datePart=$(echo ${archivePath} | awk -F_ '{print $3}')
680    timePart=$(echo ${archivePath} | awk -F_ '{print $4}')
681    archiveSuffix="${datePart}_${timePart}"
682
683    if promptForConfirmation "You are about to restore from the '${archiveDirectory}'"
684      backup set.\nYour existing data will be lost if not backed up first."; then
685      volumes=$(dockerCompose config --volumes)
686      for volume in ${volumes}; do
687        volume=$(echo ${volume} | sed 's~\r$~~')
688        targetVolume=${COMPOSE_PROJECT_NAME}_${volume}
689        archiveName=${targetVolume}_${archiveSuffix}
690        archivePath="${archiveDirectory}/${archiveName}"
691
692        restoreArchive -q "${archivePath}" "${targetVolume}"
693        done
694      else
695        echo -e \\n"Restore aborted."
696      fi
697    )
698
699 function restoreArchive()
700 {
701  (
702    local OPTIND
703    local quiet
704    unset quiet
705    while getopts q FLAG; do
706      case $FLAG in
707        q ) quiet=1 ;;
708        esac
709    done
710    shift $((OPTIND-1))
711
712    archive=${1}
713    volume=${2}
714    tarOptions=${3} # Example "--strip=1", to remove the first directory level.
715    if [ -z ${archive} ] || [ -z ${volume} ]; then
716      echoYellow "You must supply the path to the archive and the name of the volume
717      to which the archive will be restored."
718      exit 1
719    fi

```

```

720 archiveFolder=${archive%/*}
721 archiveName=${archive##*/}
722 archiveToRestore=/${ROOT_BACKUP_DIR}/${archiveName}
723 archiveVolumeMount=$(getVolumeMount ${archiveFolder})

724
725 if [ ! -z "${quiet}" ] || promptForConfirmation "You are about to restore '${archive}' to ${volume}.\\nYour existing data will be lost if not backed up first." ; then
726 deleteVolume ${volume}
727 echoYellow \\n"Restoring ${volume} from ${archive} ..."
728 docker run \
729   --rm \
730   --name von-network-restore \
731   --user root \
732   -v ${archiveVolumeMount}:/${ROOT_BACKUP_DIR} \
733   -v ${volume}:/target_volume von-network-base \
734   tar --same-owner -xvpf ${archiveToRestore} -C /target_volume/ ${tarOptions}
735 else
736   echo -e \\n"Restore aborted."
737 fi
738 )
739 }
740
741 function debugVolume()
742 {
743 (
744 volume=${1}
745 volumeMountFolder=${2:-/debug_volume}
746 if [ -z ${volume} ]; then
747   echoYellow "You must supply the name of the volume to attach to the debug session."
748   exit 1
749 fi
750
751 backupVolumeMount=$(getVolumeMount ./${ROOT_BACKUP_DIR})/
752
753 echo -e "\\nOpening a debug session with the following volume mounts:\\n - '${volume}':'${volumeMountFolder}'\\n - '${backupVolumeMount}':'/${ROOT_BACKUP_DIR}'\\n"
754 docker run \
755   --rm \
756   -it \
757   --network="host" \
758   --user root \
759   -v ${backupVolumeMount}:/${ROOT_BACKUP_DIR} \
760   -v ${volume}://${volumeMountFolder} \
761   --entrypoint ${SHELL_CMD} \
762   von-network-base
763 )
764 }
765
766 function findBackup() {
767 (
768 _fileName=${1}
769
770 # If no backup file was specified, find the most recent set.

```

```

771     # Otherwise treat the value provided as a filter to find the most recent backup
772     # set matching the filter.
773     if [ -z "${_fileName}" ]; then
774         _fileName=$(find ${ROOT_BACKUP_DIR}*-type f -printf '%T@ %p\n' | grep .tar.gz |
775             sort | tail -n 1 | sed 's~^.* \(.*\)~\1~')
776     else
777         _fileName=$(find ${ROOT_BACKUP_DIR}*-type f -printf '%T@ %p\n' | grep .tar.gz |
778             grep ${_fileName} | sort | tail -n 1 | sed 's~^.* \(.*\)~\1~')
779     fi
780
781     echo "${_fileName}"
782 }
783
784 # OSX ships with an old version of Bash that does not support the \e escape character.
785 function echoYellow () {
786     (
787         _msg=${1}
788         _yellow='\x1B[33m'
789         _nc='\x1B[0m' # No Color
790         echo -e "${_yellow}${_msg}${_nc}" >&2
791     )
792 }
793
794 function promptForConfirmation() {
795     (
796         _msg=${@}
797         echoYellow "\n${_msg}"
798         read -n1 -s -r -p $'\x1B[33mWould you like to continue?\x1B[0m Press \'Y\' to
799             continue, or any other key to exit ...\\n' key
800         if [[ $(toLowerCase ${key}) == 'y' ]]; then
801             return 0
802         else
803             return 1
804         fi
805     )
806 }
807
808 function deleteVolume() {
809     (
810         volume=${1}
811
812         echoYellow \\n"Deleting volume '${volume}' ..."
813         containerId=$(docker volume rm ${volume} 2>&1 >/dev/null | sed -e 's~.*\[\\(.*)\\]
814             \]~\1~' | grep -v ${volume})
815         if [ ! -z "${containerId}" ]; then
816             # The volume is in use by a container. Remove the container before deleting the
817             # volume.
818             docker stop ${containerId} > /dev/null 2>&1
819             docker rm ${containerId} > /dev/null 2>&1
820             docker volume rm ${volume} > /dev/null 2>&1
821         fi
822     )
823 }
824
825 function deleteVolumes() {
826     (

```

```

822     _ projectName=${COMPOSE_PROJECT_NAME:-docker}
823
824     echoYellow \\n"Stopping and removing any running containers ..."
825     ${dockerCompose} down -v
826
827     _ pattern="^${_ projectName}_\|^docker_"
828     _ volumes=$(docker volume ls -q | grep ${_ pattern})
829     if [ ! -z "${_ volumes}" ]; then
830         echoYellow "Removing project volumes ..."
831         echo ${_ volumes} | xargs docker volume rm
832     fi
833 }
834
835
836 function getVolumeMount() {
837     path=${1}
838     path=$(realpath ${path})
839     path=${path%%+(/)}
840     if [[ "$OSTYPE" == "msys" ]]; then
841         # When running on Windows, you need to prefix the path with an extra '/'
842         path="/${path}"
843     fi
844     echo ${path}
845 }
846
847 function checkFolderPermissions() {
848     # Create the tmp folder if it does not exist ...
849     if [ ! -d "${TMP_FOLDER}" ]; then
850         echo "Creating ${TMP_FOLDER} ..."
851         mkdir -p tmp
852     fi
853
854     # Ensure folder permissions are set correctly for use inside the docker container
855     ...
856     setFolderReadWriteAll "${TMP_FOLDER}"
857     setFolderReadWriteAll "${CLI_SCRIPTS_FOLDER}"
858 }
859
860 function setFolderReadWriteAll() {
861     # This has no impact on Windows. The chmod command will run but the permissions don't actually change.
862     if [[ "$OSTYPE" != "msys" ]]; then
863         folder=${1}
864         permissions=$(stat ${STAT_OPS} ${folder})
865
866         if [[ "${permissions:0-1}" != 7 ]]; then
867             echo "Setting ${folder} to read/write for all users ..."
868             chmod a+rws ${folder}
869         fi
870     fi
871 }
872
873 function isUsingWSL() {
874     kernelVersion=$(docker -l error info 2> /dev/null | grep "Kernel Version")
875     if [[ "$OSTYPE" == "msys" ]] && [[ "${kernelVersion}" == *'WSL'* ]]; then
876         return 0
877     else

```

```

877         return 1
878     fi
879 }
880
881 function getContainers() {
882     initEnv "$@"
883     ${dockerCompose} \
884     ps --all | tail -n +2 | awk '{ print $1 }'
885 }
886
887 varDockerFolder="/var/lib/docker"
888 varDockerFolderMsys="//wsl$/docker-desktop-data/version-pack-data/community/docker"
889 function truncateLogs() {
890
891     if [[ "$OSTYPE" == "msys" ]] && ! isUsingWSL ; then
892         echoYellow "Currently, the truncateLogs function is only supported in Windows when
893             using Docker on WSL"
894         exit 1
895     fi
896
897     containers=$(getContainers)
898     for container in ${containers}; do
899         log=$(docker inspect -f '{{.LogPath}}' ${container} 2> /dev/null)
900
901         if [[ "$OSTYPE" == "msys" ]]; then
902             log="${log}/${varDockerFolder}/${varDockerFolderMsys}"
903         fi
904
905         if [ -f "${log}" ]; then
906             echo "Truncating log for ${container}: ${log}"
907             truncate -s 0 ${log}
908         else
909             echoYellow "Unable to locate log for ${container}: ${log}"
910         fi
911     done
912 }
913
914 # =====
915
916 pushd "${SCRIPT_HOME}" >/dev/null
917 checkFolderPermissions
918 COMMAND=$(toLowerCase ${1})
919 shift || COMMAND=usage
920
921 case "${COMMAND}" in
922     start|up)
923         initEnv "$@"
924         export LEDGER_SEED=${LEDGER_SEED:-0000000000000000000000000000Trustee1}
925         install_taa
926         ${dockerCompose} \
927             up \
928             -d webserver node1 node2 node3 node4 pqc-sidecarproxy-webserver
929         wait_for_ledger
930         logs
931         echo 'Want to see the scrolling container logs? Run "./manage logs"'
932     ;;
933     start-combined)
934         initEnv "$@"

```

```

933     install_taa
934     ${dockerCompose} \
935         up \
936         -d webserver nodes pqc-sidecarproxy-webserver
937         wait_for_ledger
938         logs
939     ;;
940     start-web)
941         initEnv "$@"
942         ${dockerCompose} \
943             up \
944             -d webserver pqc-sidecarproxy-webserver
945             wait_for_ledger
946             logs webserver
947     ;;
948     synctest)
949         initEnv "$@"
950         ${dockerCompose} \
951             up \
952             -d synctest node1 node2 node3 node4
953             logs -f synctest
954     ;;
955     cli)
956         runCliCommand $@
957     ;;
958     indy-cli)
959         runCliCommand indy-cli $@
960     ;;
961     logs)
962         initEnv "$@"
963         logs -f "$@"
964     ;;
965     stop)
966         initEnv "$@"
967         ${dockerCompose} \
968             stop
969             remove_taa
970     ;;
971     down|rm)
972         initEnv "$@"
973         deleteVolumes
974         remove_taa
975     ;;
976     build)
977         docker build ${initDockerBuildArgs} -t von-network-base .
978     ;;
979     rebuild)
980         docker build --no-cache --progress plain ${initDockerBuildArgs} -t von-network-
981             base .
982     ;;
983     dockerhost)
984         echo -e \\n"DockerHost: ${DOCKERHOST}"\\n
985     ;;
986     generatesecrets)
987         generateSecrets
988     ;;
989     generatedid)

```

```

989         generateDid $@
990     ;;
991     generategenesisfiles)
992         trustee_csv="${1}"
993         steward_csv="${2}"
994         generateGenesisFiles "${trustee_csv}" "${steward_csv}"
995     ;;
996     apply-taa)
997         apply-taa $@
998     ;;
999
1000    backup)
1001        backup "$@"
1002    ;;
1003    restore)
1004        restore $@
1005    ;;
1006    restorearchive)
1007        archive=${1}
1008        volume=${2}
1009        tarOptions=${3}
1010        restoreArchive ${archive} ${volume} ${tarOptions}
1011    ;;
1012    debugvolume)
1013        volume=${1}
1014        volumeMountFolder=${2}
1015        debugVolume ${volume} ${volumeMountFolder}
1016    ;;
1017    truncatelogs)
1018        isUsingWSL
1019        truncateLogs
1020    ;;
1021    *)
1022        usage;;
1023 esac
1024
1025 popd >/dev/null

```

Listing A-11

indy-tails-server manage Skript

```

1 #!/bin/bash
2
3 # =====
4 # Check Docker Compose
5 # =====
6
7 # Default to deprecated V1 'docker-compose'.
8 dockerCompose="docker-compose --log-level ERROR"
9
10 # Prefer 'docker compose' V2 if available
11 if [[ $(docker compose version 2>/dev/null) == 'Docker Compose'* ]]; then
12     dockerCompose="docker --log-level error compose"
13 fi
14 export MSYS_NO_PATHCONV=1

```

```

15 # getDockerHost; for details refer to https://github.com/bcgov/DITP-DevOps/tree/main/
16   code/snippets#getdockerhost
17 . /dev/stdin <<<$(cat <(curl -s --raw https://raw.githubusercontent.com/bcgov/DITP-
18   DevOps/main/code/snippets/getDockerHost))"
19 export DOCKERHOST=$(getDockerHost)
20 set -e
21
22 # =====
23 # Usage:
24 # =====
25 usage() {
26   cat <<-EOF
27     Usage: $0 [command] [options]
28   Commands:
29     build      - Build the tails-server docker images
30     start | up - Run tails-server
31     logs       - To tail the logs of running containers (ctrl-c to exit).
32     stop | down - Stop tails-server
33     rm         - Stop tails-server and remove volumes
34 EOF
35   exit 1
36 }
37
38 toLower() {
39   echo $(echo ${@} | tr '[[:upper:]]' '[[:lower:]]')
40 }
41
42 exportEnvironment() {
43   for arg in "$@"; do
44     case "$arg" in
45       *=*)
46       export "${arg}"
47       ;;
48     --logs)
49       TAIL_LOGS=1
50       ;;
51     *)
52       # If not recognized, save it for later processing ...
53       set -- "$@" "$arg"
54       ;;
55     esac
56   done
57   export GENESIS_URL=${GENESIS_URL:-http://$DOCKERHOST:9000/genesis}
58   export STORAGE_PATH=${STORAGE_PATH:-/tmp/tails-files}
59   export LOG_LEVEL=${LOG_LEVEL:-info}
60   export LOGGING_CONFIG=${LOGGING_CONFIG:-/tails_server/config/logging-config.yml}
61   export TAILS_SERVER_URL=${TAILS_SERVER_URL:-http://$DOCKERHOST:6543}
62 }
63
64 function logs() {
65 (
66   local OPTIND
67   local unset _force
68   local unset no_tail
69   while getopts ":f:" FLAG; do

```

```

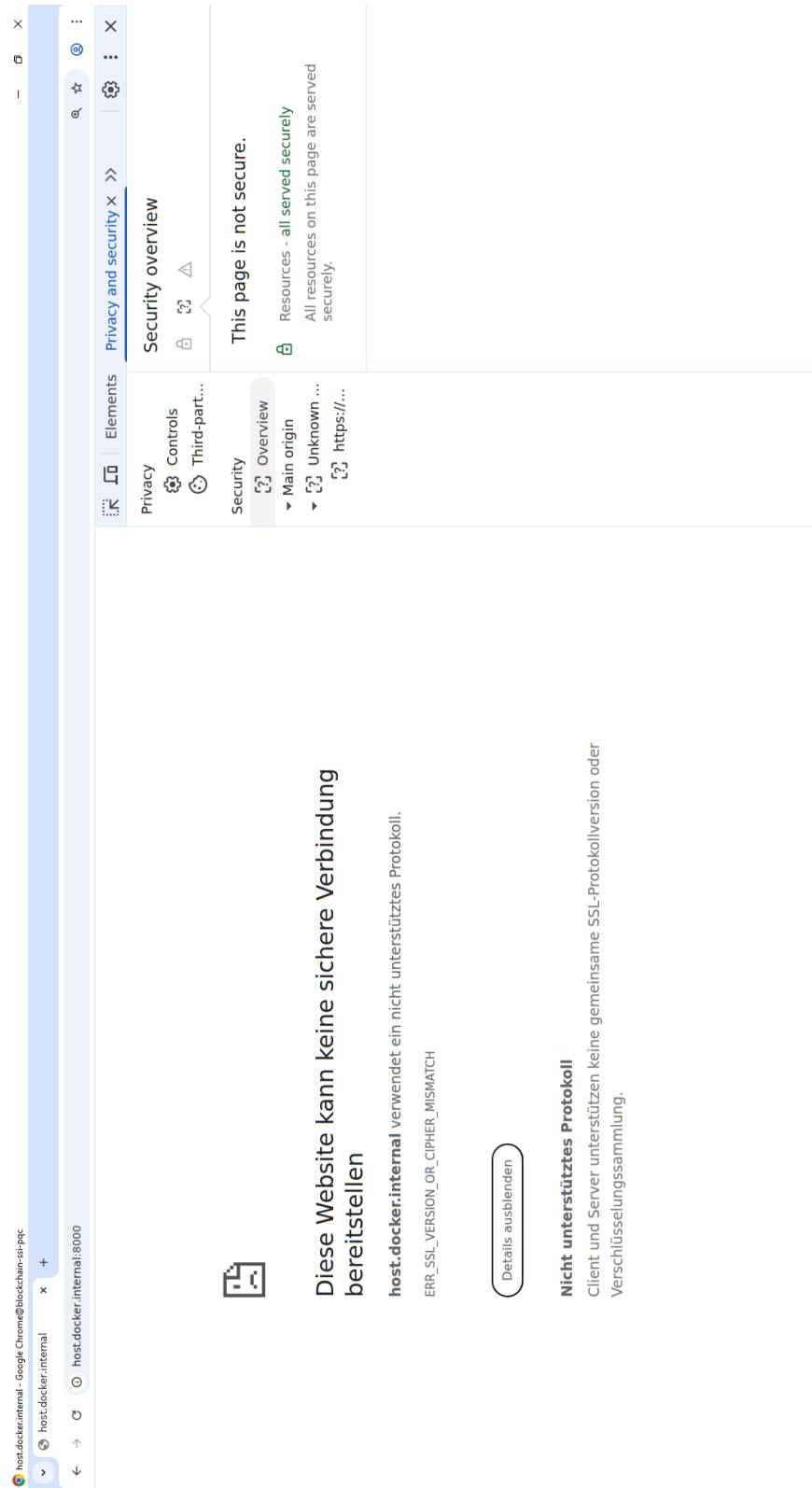
70      case $FLAG in
71          f ) local _force=1 ;;
72          - )
73              case ${OPTARG} in
74                  "no-tail") no_tail=1
75                  ;;
76              esac
77          esac
78      done
79      shift $((OPTIND-1))
80
81      log_args=()
82      (( no_tail != 1 )) && log_args+=(-f)
83      if [ ! -z "${TAILS_LOGS}" ] || [ ! -z "${_force}" ]; then
84          ${dockerCompose} logs \
85          "${log_args[@]}" "$@"
86      fi
87  )
88 }
89
90 # =====
91 pushd "${SCRIPT_HOME}" >/dev/null
92 COMMAND=$(toLowerCase ${1})
93 shift || COMMAND=usage
94
95 case "${COMMAND}" in
96 build)
97     ${dockerCompose} build @@
98     ;;
99 start|up)
100    exportEnvironment @@
101    ${dockerCompose} up -d tails-server pqc-sidecarproxy-tails-server
102    logs
103    echo "Run './manage logs' for logs"
104    ;;
105 test)
106    exportEnvironment @@
107    ${dockerCompose} up -d ngrok-tails-server tails-server
108    ${dockerCompose} run tester --genesis-url $GENESIS_URL --tails-server-url
109        $TAILS_SERVER_URL
110    # ${dockerCompose} down
111    ;;
112 logs)
113    ${dockerCompose} logs -f
114    ;;
115 stop)
116    ${dockerCompose} stop
117    ;;
118 down|rm)
119    ${dockerCompose} down -v
120    ;;
121 *)
122     usage
123     ;;
124 esac
popd >/dev/null

```

Anhang 3.3: Formative Evaluation

Anhang 3.3.1: Cipher Mismatch der TLS-1.3-Verbindung des Webservers

Abbildung A-7
Cipher Mismatch der TLS-1.3-Verbindung des Webservers



Anmerkung. Eigene Darstellung.

Anhang 3.3.2: Eigenkomplilation eines PQC-Chromium-Browsers

Zur experimentellen Evaluation von Verfahren der PQC im Kontext realer Webbrowser wurde ein Chromium-basierter Browser mit erweiterten TLS-Fähigkeiten selbst kompiliert. Grundlage bildete das von OQS bereitgestellte Chromium-Demoprojekt, das eine Integration der *liboqs*-Bibliothek in die TLS-Implementierung von Chromium (BoringSSL) demonstriert und hybride sowie rein PQ-basierte Schlüsselaustausch- und Signaturverfahren bereitstellt (open-quantum-safe, n. d. a, n. d. b). Ziel war es, eine lauffähige Build-Umgebung unter Linux aufzusetzen, den OQS-anangepassten Quellcode zu beziehen, die notwendigen Abhängigkeiten zu installieren und anschließend ein reproduzierbares Build-Artefakt des Browsers mit PQC-Unterstützung zu erzeugen.

Die Einrichtung der Build-Umgebung erfolgte weitgehend entsprechend der offiziellen Linux-Build-Dokumentation des Chromium-Projekts („Chromium Docs - Checking out and Building Chromium on Linux“, n. d.). Hierzu wurden zunächst die von Chromium bereitgestellten *depot_tools* geklont und in den PATH eingebunden, um Werkzeuge wie *fetch*, *gclient* und *gn* verwenden zu können. Anschließend wurde ein separates Arbeitsverzeichnis angelegt und über *fetch* ein Chromium-Checkout inklusive aller benötigten Abhängigkeiten durchgeführt. Unter einer aktuellen Ubuntu-Linux-Distribution wurden im nächsten Schritt die von Chromium empfohlenen Systemabhängigkeiten installiert, etwa über das Skript „*build/install-build-deps.sh*“, welches Compiler, Entwicklungsbibliotheken und Laufzeitbibliotheken für das spätere Linken der Browser-Binärdateien bereitstellt („Chromium Docs - Checking out and Building Chromium on Linux“, n. d.). Nach Abschluss dieser Vorbereitungen wurden mittels „*gclient runhooks*“ die Chromium-spezifischen Hooks ausgeführt, um zusätzliche Werkzeuge und vorkompilierte Komponenten nachzuladen.

Aufbauend auf dieser Standard-Umgebung wurde der von OQS bereitgestellte Chromium-Zweig eingebunden, der Anpassungen an BoringSSL sowie die Einbindung von *liboqs* enthält (open-quantum-safe, n. d. a, n. d. b). Dazu wurde das entsprechende Repository aus dem OQS-Demoprojekt geklont und gemäß der dort beschriebenen Struktur so in die bestehende Chromium-Arbeitsumgebung integriert, dass die PQC-Erweiterungen anstelle der unveränderten Upstream-Kryptografiebibliothek verwendet werden. Zentral war dabei die Übernahme der in der OQS-Dokumentation beschriebenen Build-Konfigurationen, insbesondere GN-Argumente, die das Linken gegen *liboqs* aktivieren und die experimentellen PQ- bzw. Hybrid-Ciphersuites in der TLS-Konfiguration von Chromium einschalten (open-quantum-safe, n. d. b). Diese Konfiguration wurde in einer eigenen Build-Directory, etwa „*out/oqs-Default*“, über den Aufruf „*gn gen*“ mit den projektspezifischen Argumenten erzeugt.

Im Anschluss daran erfolgte der eigentliche Kompilervorgang des Browsers mit dem von Chromium vorgesehenen Build-Werkzeug autoninja, das die GN-Konfiguration nutzt, um alle notwendigen Targets effizient zu bauen („Chromium Docs - Checking out and Building Chromium on Linux“, n. d.). Durch den Aufruf von „autoninja -C out/oqs-Default chrome“ wurde eine Browser-Binärdatei erzeugt, die die OQS-Erweiterungen in der TLS-Schicht enthält. Der resultierende Browser konnte direkt aus dem Build-Verzeichnis gestartet und gegen PQC-fähige Testserver genutzt werden, um TLS-Verbindungen mit hybriden oder rein PQ-basierten Schlüsselaustauschmechanismen zu etablieren. Die so aufgebaute Umgebung ermöglicht eine kontrollierte experimentelle Analyse der praktischen Auswirkungen von PQC im Browserkontext, etwa hinsichtlich Kompatibilität, Performance und Protokollhandshake, auf Basis eines realen Chromium-Builds mit explizit aktiverter PQC.

Anhang 3.3.3: Validierung der Zertifikatskette und ML-DSA-Signaturen

Abbildung A-8*Erfolgreiche Validierung des ML-DSA-Zertifikats des Issuers*


```
ferris@blockchain-sst-poc: ~
ferris@blockchain-sst-poc: $ openssl s_client -connect host.docker.internal:8021
Connecting to 172.17.0.1
CONNECTED(00000003)
depth=0 CN=pqc reverseproxy issuer agent, O=FM, C=DE
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN=pqc reverseproxy issuer agent, O=FM, C=DE
verify error:num=21:unable to verify the first certificate
verify return:1
depth=0 CN=pqc reverseproxy issuer agent, O=FM, C=DE
verify return:1
...
-----  

Certificate chain  

  0 s:CN=pqc reverseproxy issuer agent, O=FM, C=DE  

    i:CN=Master Thesis PQC Root CA FM, O=FM, C=DE  

      a:KEY: ML-DSA-05, 15616 (bit); sigalg: ML-DSA-87  

      v:NotBefore: Oct 23 18:01:00 2025 GMT; NotAfter: Oct 23 18:03:00 2026 GMT  

...
-----  

Server certificate  

-----BEGIN CERTIFICATE-----  

MIIBRzCCRGawIBAgJUNoGRVjYUv7CMgJXjdULBLh7AwMwYJYIZIAmUDAnT  

MEExJTAjBgNVBAgNMHElh3RlcvBiaGUzaxMgUFDFJvb3Qg90EgRk0xczABgNV  

BiomAKzNpm0swCQDVQDGEWEHrTAf-w0yNTEmWjmxODAzhBaw0yNjEWm1mXODA  

MDBaMEtxJxJAKBgvNBAMMHx8xxYByZXZ1.cnNichiJveHkgAXNzamWVJGFfn2M50HQw  

COVDQKDAgITTELMAg1UEBHMREungyMAg0WCgSAFLwQD5gOCB4EAwLxF  

tJuZvdpT3ah7Sa139awl_j69QH0SuvEm31M+vwCMQs06qaj21Wb0k1ghpx6VnPnH  

99gt8p+fL2vBqj09BKRLQhMoX765xh62zIoulksra7q25xsa17rkszr55pbnp01  

41PNjAT3ApplYcSEv1-k77FowZn6bsICh38uidiGmH+tyQNMW6rdDiG69wunbIJ5-MZ  

1sTsmXr+FJN1Stgcl-2vGEWotu48vulck5z2-fJrD1++czpor1gQemuUd3b749t51  

0stICD6sBqabeyef09qjNEad54tj8q8Bu7Pn1Hxj/viuChM/noy0ij5KM9ujKphxN  

2azuf5jubFFVz9ubgoZ0NT1x2L8xteHgXHAQhA9DUkZ2Vj1+e80Hds14b4mtuI  

vx88/yFjTlxXp9unur91BfxjY3hiCY2KG3t/vio/TyE/bG2cbca9bQ7Em5xJkvc  

2i+Cog1zVTDnEGEB9cnPvc3metCh1mc1nTchc1nUGJ0X1n5DO1Lfut+hdfv  

mRtQGEPtWmNa03vion0h8vQ/c55YtUs09fExfQushHgqaAf4h0t6OYfwaQwnkQT  

YhjVkrjk2Vd00A2.i./n9vqeD5SQzzLHSjL./WqkkrBhf6P2peacgHBqav76yTT8BW  

Lrx5dFnC-3ukNvp2E929Qunh711CREMfEImKE512tKyv1C01V1YKiuotpdbP0h  

y5aiyN54hBvkhb5C5afCTP21HyvR3LSNzMi1uNz4pdhs51qAmsA+dXkyA6R72mW  

78jKKkBv8wIEqkba2nllaaguz15uJ293NBFiloiqDAPFSpxs7/Pz2mV7+vepakZ04B  

5ZQjVpDTPnvuky9tBwBLDgyrTo3SSsNdPvFRdkocXbc3u0mEZGEhLNniyEV1+Q0
```

Anmerkung. Eigene Darstellung.

Anhang 3.3.4: Validierung der TLS 1.3 Algorithmen-Aushandlung

Abbildung A-9

Erfolgreiche Validierung der TLS-1.3-Verbindung des Issuers

```

fernsehblockchain-test-poc: ~
× + ×
○ 7hw5AYINDsphNVMJ2GTEJvU0An4Vi5whxCmVNUYt0TFM096fp8gUMIVUWyg/ou
EF/nupITHysF130RMmiUNurt05V1SADCm865j1vaDbh40kwYpprIqerUTKri650wMpZ
mIxrg7jN13lUgMnhYxkxo02iJuVj9wtX8xuqhiffe08HC7OM3j3jkURWBSVAU1LabL
GfeyX-sdg5-wJmaVsbhMjB3o3ER1Ad/kswMuCgtz8wxX25s9hiHTVgK2ztf8HR4r
PhhWd6D5B9LTTuN9CTViazZ9osrvW20EhjLTDashkoOcmhblH1+9NkbDEz\rv
ia2yxcc38AAAAAAAACAAAAAAAUFPyInMTY+
---END CERTIFICATE---
subject=CN=pqc reverseproxy issuer agent, O=FM, C=DE
issuer=CN=Master Thesis PQC Root CA FM, O=FM, C=DE
No client certificate CA names sent
Peer signature type: mdsas55
Negotiated TLS1.3 group: X2519MLKEN768
SSL handshake has read 11688 bytes and written 1640 bytes
Verification error: unable to verify the first certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Protocol: TLSv1.3
Server public key is 15616 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 21 (unable to verify the first certificate)
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
Protocol : TLSv1.3
Cipher   : TLS_AES_256_GCM_SHA384
Session-ID: EBB785B9FDA2F72D50DD3B1BCE0200BA21CA12F2B6262073A95F93EB7B2983B
Session-ID-ctx:
Resumption PSK: 38FC6318D148CAA70CDAB2B96BD77929B28EF0D7AB2FD4D9DBB560C7B2FD4D8CD9C0DE53BC1C0999F2A267DC5264EB6F262
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 300 (seconds)

```

Anmerkung. Eigene Darstellung.

Anhang 3.3.5: Validierung der Ledger-Initialisierung

Abbildung A-10*Erfolgreiche Validierung der TLS-1.3-Verbindung des Webservers*

The screenshot shows a web browser interface with the following details:

- Address Bar:** localhost:8000 -> localhost Indy Network + host.docker.internal:8000
- Security Overview:**
 - This page is secure (valid HTTPS).
 - Certificate - valid and trusted: The connection to this site is using a valid, trusted server certificate issued by Master Thesis POC Root CA Fm.
 - View certificate
- Content Area:**
 - Connect to the Network:** Download the genesis transaction file to connect to the network. JSON Genesis Transaction
 - Authenticate a New DID:** Easily write a new DID to the ledger for new identity owners. Radio buttons: Register from seed (selected) or Register from DID. Wallet seed (32 characters or base64) input field.
 - Validator Node Status:**
 - Node1:** DID: GwfpOLhBcoqesN7zqfotIgFa7cbuqdZpkx3Xo6plhp hv Uptime: 23 hours, 10 minutes, 15 seconds Txns: 0 config, 5 ledger, 4 pool, 0.0154/s read, 0/s write indy-node version: 1.12.6
 - Node2:** DID: 8ECSVSk179mj5jKRBLWiqssMLgq6EPmXtAty5tWPSG Ab Uptime: 23 hours, 10 minutes, 15 seconds Txns: 0 config, 5 ledger, 4 pool, 0.0153/s read, 0/s write indy-node version: 1.12.6
 - Node3:** DID: DKVxG2fXXTUb7SN7hGEfXB3dfdAnYv11czDfUHpmDxya Uptime: 23 hours, 10 minutes, 15 seconds Txns: 0 config, 5 ledger, 4 pool, 0.0149/s read, 0/s write indy-node version: 1.12.6
 - Node4:** DID: 4P3EDQ2dW1tc11Bp6543cfuebJFg936kLAucskGf aA Uptime: 23 hours, 10 minutes, 15 seconds Txns: 0 config, 5 ledger, 4 pool, 0.0152/s read, 0/s write indy-node version: 1.12.6

Bottom Right: View detailed information about the status of the running validator nodes. ▶ Detailed Status

Anmerkung. Eigene Darstellung.

Abbildung A-11

Erfolgreiche Validierung der Genesis-Datei des Webservers

```
ferris@blockchain-sssi-pqc:~$ curl https://host.docker.internal:8000/genesis
{
  "reqSignature": {},
  "txns": [
    {
      "alias": "Node1",
      "blskey": "4hRaUNHISgjQVgkpm8nhNEFd6txHznoYReg9krmJjkivg4oSEimFF6ns06M410vhM2233hvses5v
f5N9n1uWmJBYrtWWhYMA7n76vlu13zU8BkyeA7chFsfh3hAgdUHCxcsEchHz6jncyz1P2U2nzb2DvTuL5wkpheoBfBaI KmBa",
      "data": {
        "ip": "192.168.100.2",
        "port": "9701"
      },
      "node_ip": "192.168.100.2",
      "node_port": "9701",
      "services": [
        "VALIDATOR"
      ],
      "txmMetadata": {
        "seqNo": 1,
        "txnId": "Fee82e10e894f197e2bea7d6296a6d6f50f93f8eed954ec61b2ed2950b823"
      }
    },
    {
      "alias": "Node2",
      "blskey": "37T4APPxVoxZkhZ7dg9kule52yXryu1X0Mp6LbwDB7L5b662Lsb33s5G7zq58T1K1MNxwuChj1FKNzV
psnfMq1G1vXh88rt38mNF59tNe0Hd2Bz5yCuoBnPrpVYD909TNaPadvRvqkLByCabub7z3XXkbBeShzpuMa5QyPj0jk",
      "data": {
        "ip": "192.168.100.2",
        "port": "9702"
      },
      "node_ip": "192.168.100.2",
      "node_port": "9702",
      "services": [
        "VALIDATOR"
      ],
      "txmMetadata": {
        "seqNo": 2,
        "txnId": "1ac8aae2a18ceu660-fef8694b61"
      }
    },
    {
      "alias": "Node3",
      "blskey": "3WFpd9g7C5cnLYzWfZevJqhubkFLAbFccBok15GdrkMNUUJgsk3jv60Kj617gEubf70qCaFxNdkm7
eswo4lsdkTRc2tLGzBdavNqj8dwpzuprluUF22KTHTPQbium8Y4QFYEF2u5u27JcNkdgyPeUSX12u5lqdD0nNSWUK5da5",
      "data": {
        "ip": "192.168.100.2",
        "port": "9703"
      },
      "node_ip": "192.168.100.2",
      "node_port": "9703",
      "services": [
        "VALIDATOR"
      ],
      "txmMetadata": {
        "seqNo": 3,
        "txnId": "7af9f355df7a78ed24668f0a0e369aacaaf08ba8773e3026a1608ba85f35fc"
      }
    },
    {
      "alias": "Node4",
      "blskey": "2Zn3bHm1m4fL254HJIVSwvq2rChVp8jklwsweLAEVJCVx6Mm1WnQd15RPyM5UD72vwvhue6WARK3
KxWeimsanSmvIVkRabeBEhxedeayjzJFGPydyev1lxqXhmtVAnBkOpPydu7TAox5f77NnRAdeLmlu99GEru77D8kfAa6hlp09aw",
      "data": {
        "ip": "192.168.100.2",
        "port": "9704"
      },
      "node_ip": "192.168.100.2",
      "node_port": "9704",
      "services": [
        "VALIDATOR"
      ],
      "txmMetadata": {
        "seqNo": 4,
        "txnId": "aa5e817d7c626170eca17552202
      }
    }
  ]
}
```

Anmerkung. Eigene Darstellung.

Anhang 3.3.6: Validierung der ACA-Py API-Verfügbarkeit

Listing A-12

Issuer Agent Boot Logs

```

1 Executing task in folder ferris: docker logs --tail 1000 -f 179
   e43b336fa16b399efa7326cdcc0b8bfa6ab24c8f86a2b4b630fe57fc382064
2
3 2025-11-28 23:49:38,921 acapy_agent.config.default_context INFO Registering default
   plugins
4 2025-11-28 23:49:39,083 acapy_agent.config.default_context INFO Registering askar
   plugins
5 2025-11-28 23:49:39,308 acapy_agent.config.ledger INFO Fetching genesis transactions
   from: https://host.docker.internal:8000/genesis
6 2025-11-28 23:49:46,340 acapy_agent.core.profile INFO Create profile manager: askar
7 2025-11-28 23:49:46,827 acapy_agent.config.wallet INFO Created new profile - Profile
   name: issuer_wallet, backend: askar
8 2025-11-28 23:49:46,829 acapy_agent.config.wallet INFO No public DID created
9 2025-11-28 23:49:46,885 acapy_agent.config.ledger INFO Ledger configuration complete
10 2025-11-28 23:49:46,885 acapy_agent.core.conductor INFO Ledger configured successfully

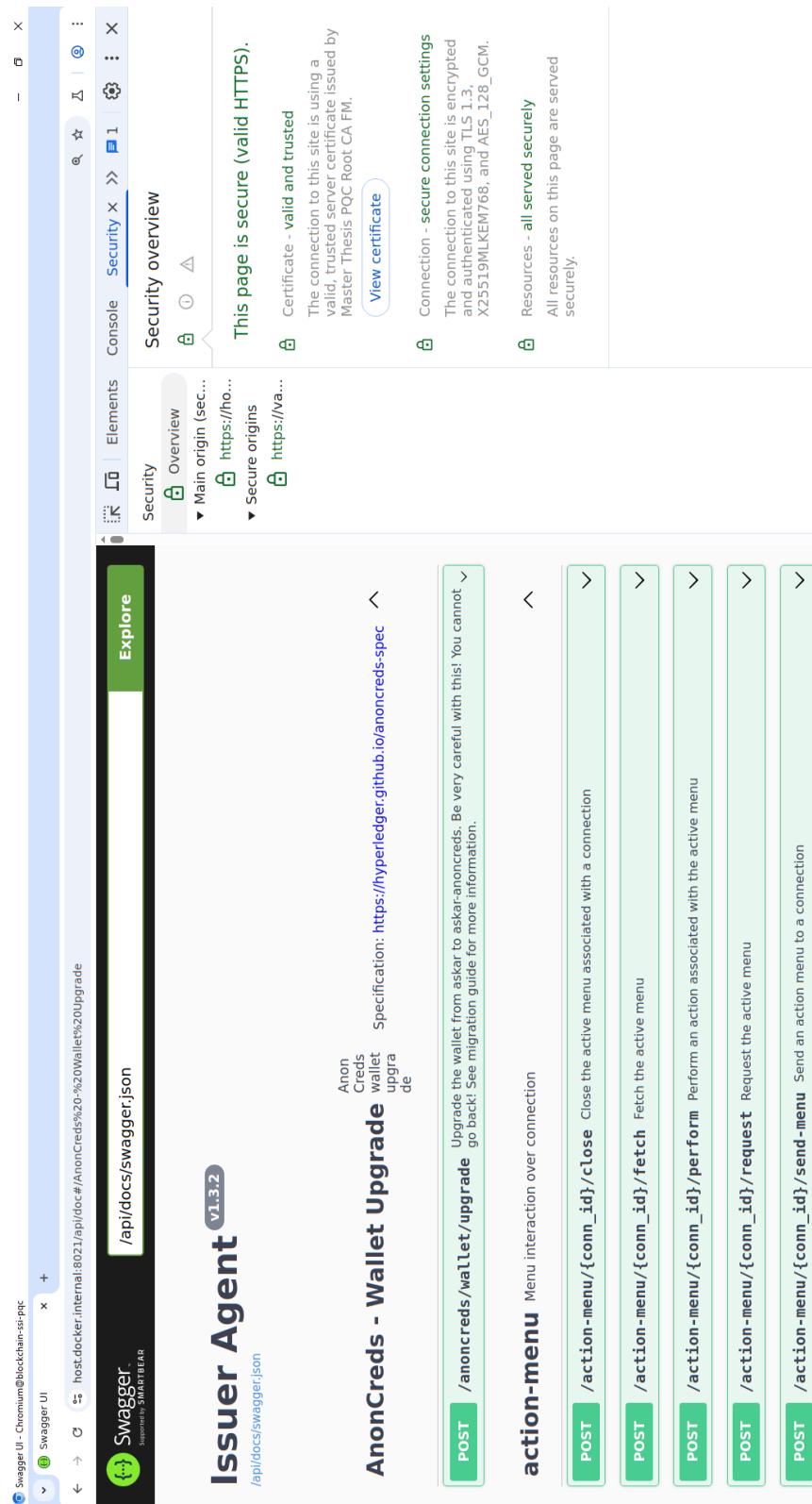
.
11 2025-11-28 23:49:46,893 acapy_agent.core.conductor INFO Wallet type record not found.
12 2025-11-28 23:49:46,894 acapy_agent.core.conductor INFO New agent. Setting wallet type
   to askar.
13 2025-11-28 23:49:47,028 acapy_agent.config.banner INFO
14 ::::::::::::::::::::
15 ::           Issuer Agent           ::
16 :::::
17 :::::
18 ::: Inbound Transports:          ::
19 :::::
20 :::   - http://0.0.0.0:8020       ::
21 :::::
22 ::: Outbound Transports:         ::
23 :::::
24 :::   - http
25 :::   - https
26 :::::
27 ::: Administration API:        ::
28 :::::
29 :::   - http://0.0.0.0:8021       ::
30 :::::
31 ::::: ver: 1.3.2 :::
32 ::::::::::::::::::::
33
34 2025-11-28 23:49:47,028 acapy_agent.config.banner INFO
35 ::::::::::::::::::::
36 ::           DEPRECATION NOTICE:   ::
37 :: -----
38 :: Receiving a core DIDComm protocol with the 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec' :::
39 :: prefix is deprecated. All parties sending this prefix should be notified that   :::
40 :: support for receiving such messages will be removed in a future release. Use   :::
41 :: https://didcomm.org/ instead.                                              :::
42 :: -----
43 :: Aries RFC 0160: Connection Protocol is deprecated and support will be removed in :::
44 :: a future release; use RFC 0023: DID Exchange instead.                         :::

```

```

45 :: -----
46 :: Aries RFC 0036: Issue Credential 1.0 is deprecated and support will be removed :: 
47 :: in a future release; use RFC 0453: Issue Credential 2.0 instead. :: 
48 :: -----
49 :: Aries RFC 0037: Present Proof 1.0 is deprecated and support will be removed in a :: 
50 :: future release; use RFC 0454: Present Proof 2.0 instead. :: 
51 ::::::::::::::::::::: 
52 
53 2025-11-28 23:49:47,029 acapy_agent.core.conductor INFO Wallet version storage record
   not found.
54 2025-11-28 23:49:47,030 acapy_agent.core.conductor INFO No upgrade from version was
   found from wallet or via --from-version startup argument. Defaulting to v0.7.5.
55 2025-11-28 23:49:47,031 acapy_agent.core.conductor INFO Upgrade configurations
   available. Initiating upgrade.
56 2025-11-28 23:49:47,033 acapy_agent.commands.upgrade INFO No ACA-Py version found in
   wallet storage.
57 2025-11-28 23:49:47,033 acapy_agent.commands.upgrade INFO Selecting v0.7.5 as --from-
   version from the config.
58 2025-11-28 23:49:47,033 acapy_agent.commands.upgrade INFO Running upgrade process for
   v0.8.1
59 2025-11-28 23:49:47,034 acapy_agent.commands.upgrade INFO No records of <class '
   acapy_agent.connections.models.conn_record.ConnRecord'> found
60 2025-11-28 23:49:47,040 acapy_agent.commands.upgrade INFO acapy_version storage record
   set to v1.3.2
61 2025-11-28 23:49:47,042 acapy_agent.core.conductor INFO Listening...
62 2025-11-28 23:49:52,008 aiohttp.access INFO 127.0.0.1 [28/Nov/2025:23:49:52 +0000] "
   GET /status/ready HTTP/1.1" 200 138 "-" "curl/7.88.1"
63 2025-11-28 23:50:22,041 aiohttp.access INFO 127.0.0.1 [28/Nov/2025:23:50:22 +0000] "
   GET /status/ready HTTP/1.1" 200 138 "-" "curl/7.88.1"

```

Abbildung A-12*Erfolgreiche Validierung der Issuer Agent ACA-Py Swagger API**Anmerkung.* Eigene Darstellung.

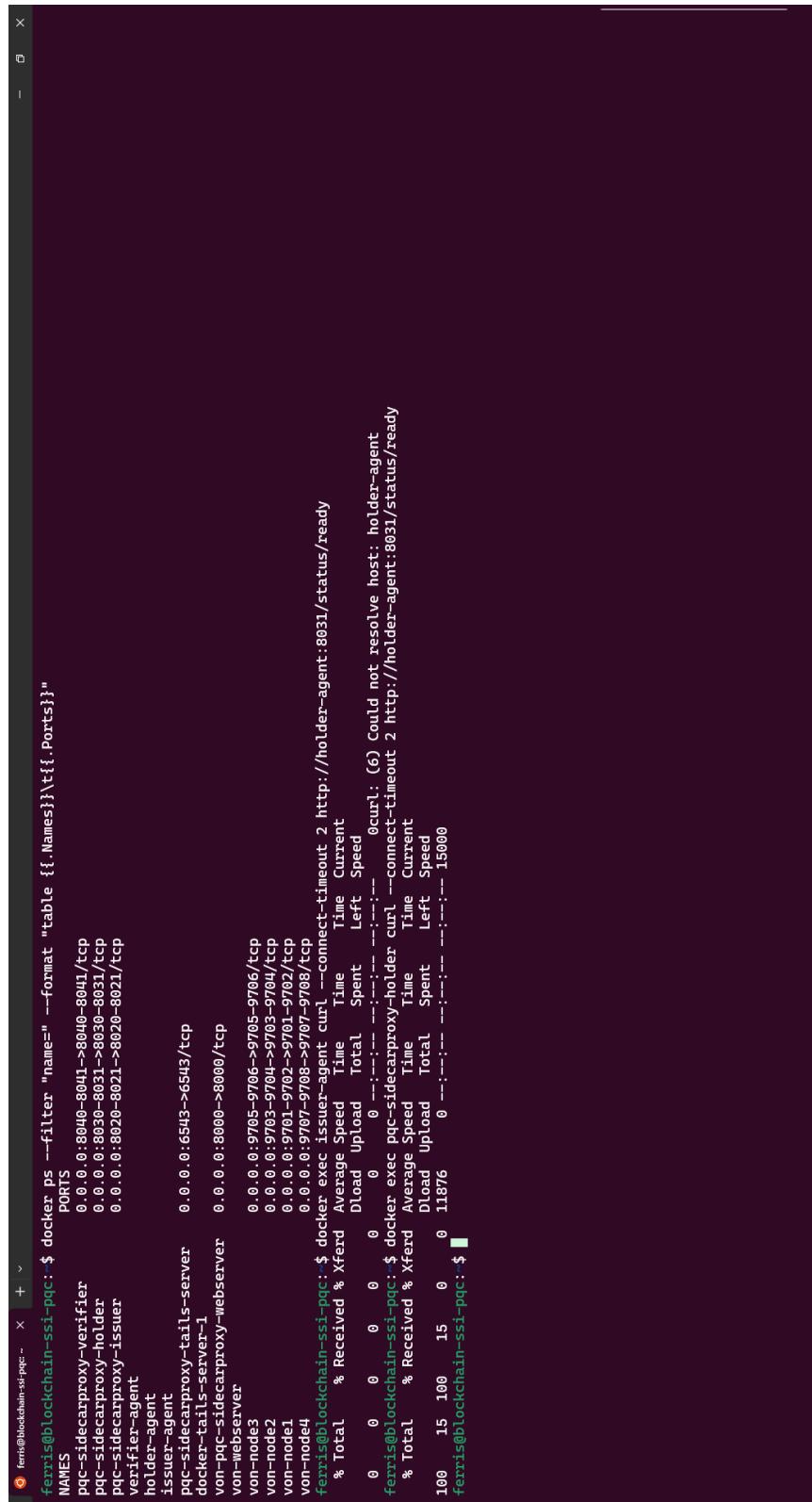
Anhang 3.3.7: Validierung der Netzwerkisolation

Abbildung A-13

Darstellung der Netzwerkisolation innerhalb der Gesamtarchitektur

Anmerkung. Eigene Darstellung.

Abbildung A-14
Erfolgreiche Validierung der Netzwerkisolation



```

ferris@blockchain-ssi-poc: ~
ferris@blockchain-ssi-poc: $ docker ps --filter "name=~ 'poc-sidecarproxy-issuer' "
PORTS
  0.0.0.0:8040-8041->8040-8041/tcp
  0.0.0.0:8030-8031->8030-8031/tcp
  0.0.0.0:8020-8021->8020-8021/tcp
  0.0.0.0:6543-6543/tcp
  0.0.0.0:8000-8000/tcp
  0.0.0.0:9705-9706->9705-9706/tcp
  0.0.0.0:9703-9704->9703-9704/tcp
  0.0.0.0:9701-9702->9701-9702/tcp
  0.0.0.0:9707-9708->9707-9708/tcp
ferris@blockchain-ssi-poc: $ docker exec issuer-agent curl --connect-timeout 2 http://holder-agent:8031/status/ready
% Total    % Received % Xferd  Average Speed   Time      Current
          0       0       0        0      0:00:00.000000000   0:00:00.000000000
          0       0       0        0      0:00:00.000000000   0:00:00.000000000
ferris@blockchain-ssi-poc: $ docker exec poc-sidecarproxy-holder curl --connect-timeout 2 http://holder-agent:8031/status/ready
% Total    % Received % Xferd  Average Speed   Time      Current
          0       0       0        0      0:00:00.000000000   0:00:00.000000000
100  15  15  0  0  11876  0  0:00:00.000000000   0:00:00.000000000
ferris@blockchain-ssi-poc: $ █

```

Anmerkung. Eigene Darstellung.

Anhang 4: Zweite Iteration der Artefaktentwicklung

Anhang 4.1: ACA-Py-Plugin-Quellcode (pqc_didpeer4_fm)

Anhang 4.1.1: liboqs_wrapper.py

Listing A-13

pqc_didpeer4_fm - liboqs_wrapper.py

```
1 """Direct liboqs-python integration for PQC key generation and operations.
2
3 This module provides a wrapper around liboqs-python for generating and using
4 ML-DSA-65 (Dilithium3) and ML-KEM-768 (Kyber768) keys.
5 """
6
7 import logging
8 from typing import Tuple, Optional
9
10 try:
11     import oqs
12 except ImportError:
13     raise ImportError(
14         "liboqs-python is required for PQC support."
15         "Install with: pip install liboqs-python>=0.10.0"
16    )
17
18 LOGGER = logging.getLogger(__name__)
19
20
21 class LibOQSWrapper:
22     """Wrapper for liboqs-python providing PQC key generation and operations."""
23
24     def __init__(self):
25         """Initialize LibOQS wrapper."""
26         self.oqs = oqs
27         LOGGER.info("LibOQS wrapper initialized")
28
29     def generate_ml_dsa_65_keypair(self) -> Tuple[bytes, bytes]:
30         """Generate ML-DSA-65 (Dilithium3) keypair.
31
32             Returns:
33                 Tuple[bytes, bytes]: (public_key, secret_key)
34         """
35
36         try:
37             sig = self.oqs.Signature("Dilithium3")
38             public_key = sig.generate_keypair()
39             secret_key = sig.export_secret_key()
40             LOGGER.debug(
41                 f"Generated ML-DSA-65 keypair: "
42                 f"public_key={len(public_key)} bytes, "
43                 f"secret_key={len(secret_key)} bytes"
44            )
45            return public_key, secret_key
46        
```

```

45         except Exception as e:
46             LOGGER.error(f"Failed to generate ML-DSA-65 keypair: {e}")
47             raise
48
49     def generate_ml_kem_768_keypair(self) -> Tuple[bytes, bytes]:
50         """Generate ML-KEM-768 (Kyber768) keypair.
51
52         Returns:
53             Tuple[bytes, bytes]: (public_key, secret_key)
54         """
55
56         try:
57             # Use KeyEncapsulation (not KEM) - correct liboqs-python API
58             kem = self.oqs.KeyEncapsulation("ML-KEM-768")
59             public_key = kem.generate_keypair()
60             secret_key = kem.export_secret_key()
61             LOGGER.debug(
62                 f"Generated ML-KEM-768 keypair: "
63                 f"public_key={len(public_key)} bytes, "
64                 f"secret_key={len(secret_key)} bytes"
65             )
66             return public_key, secret_key
67         except Exception as e:
68             LOGGER.error(f"Failed to generate ML-KEM-768 keypair: {e}")
69             raise
70
71     def sign_ml_dsa_65(self, message: bytes, secret_key: bytes) -> bytes:
72         """Sign a message using ML-DSA-65 (Dilithium3).
73
74         Args:
75             message: Message to sign
76             secret_key: ML-DSA-65 secret key
77
78         Returns:
79             bytes: Signature
80         """
81
82         try:
83             sig = self.oqs.Signature("Dilithium3", secret_key)
84             signature = sig.sign(message)
85             LOGGER.debug(f"Created ML-DSA-65 signature: {len(signature)} bytes")
86             return signature
87         except Exception as e:
88             LOGGER.error(f"Failed to sign with ML-DSA-65: {e}")
89             raise
90
91     def verify_ml_dsa_65(
92         self, message: bytes, signature: bytes, public_key: bytes
93     ) -> bool:
94         """Verify an ML-DSA-65 (Dilithium3) signature.
95
96         Args:
97             message: Original message
98             signature: Signature to verify
99             public_key: ML-DSA-65 public key
100
101        Returns:
102            bool: True if signature is valid, False otherwise
103        """

```

```

102     try:
103         sig = self.oqs.Signature("Dilithium3")
104         is_valid = sig.verify(message, signature, public_key)
105         LOGGER.debug(f"ML-DSA-65 signature verification: {is_valid}")
106         return is_valid
107     except Exception as e:
108         LOGGER.error(f"Failed to verify ML-DSA-65 signature: {e}")
109         return False
110
111     def encapsulate_ml_kem_768(self, public_key: bytes) -> Tuple[bytes, bytes]:
112         """Encapsulate a shared secret using ML-KEM-768 (Kyber768)."""
113
114         Args:
115             public_key: ML-KEM-768 public key
116
117         Returns:
118             Tuple[bytes, bytes]: (ciphertext, shared_secret)
119         """
120
121         try:
122             kem = self.oqs.KeyEncapsulation("ML-KEM-768")
123             ciphertext, shared_secret = kem.encap_secret(public_key)
124             LOGGER.debug(
125                 f"ML-KEM-768 encapsulation: "
126                 f"ciphertext={len(ciphertext)} bytes, "
127                 f"shared_secret={len(shared_secret)} bytes"
128             )
129             return ciphertext, shared_secret
130         except Exception as e:
131             LOGGER.error(f"Failed to encapsulate with ML-KEM-768: {e}")
132             raise
133
134     def decapsulate_ml_kem_768(self, ciphertext: bytes, secret_key: bytes) -> bytes:
135         """Decapsulate a shared secret using ML-KEM-768 (Kyber768)."""
136
137         Args:
138             ciphertext: Ciphertext from encapsulation
139             secret_key: ML-KEM-768 secret key
140
141         Returns:
142             bytes: Shared secret
143         """
144         try:
145             kem = self.oqs.KeyEncapsulation("ML-KEM-768", secret_key)
146             shared_secret = kem.decap_secret(ciphertext)
147             LOGGER.debug(f"ML-KEM-768 decapsulation: {len(shared_secret)} bytes")
148             return shared_secret
149         except Exception as e:
150             LOGGER.error(f"Failed to decapsulate with ML-KEM-768: {e}")
151             raise
152
153         # Convenience aliases for pqc_didcomm_v1.py compatibility
154         def kem_encapsulate(self, public_key: bytes) -> Tuple[bytes, bytes]:
155             """Alias for encapsulate_ml_kem_768()."""
156             return self.encapsulate_ml_kem_768(public_key)
157
158         def kem_decapsulate(self, secret_key: bytes, ciphertext: bytes) -> bytes:
159             """Alias for decapsulate_ml_kem_768()."""

```

```

159         return self.decapsulate_ml_kem_768(ciphertext, secret_key)
160
161     def ml_dsa_sign(self, secret_key: bytes, message: bytes) -> bytes:
162         """Alias for sign_ml_dsa_65()."""
163         return self.sign_ml_dsa_65(message, secret_key)
164
165     def ml_dsa_verify(self, public_key: bytes, message: bytes, signature: bytes) ->
166         bool:
167         """Alias for verify_ml_dsa_65()."""
168         return self.verify_ml_dsa_65(message, signature, public_key)
169
170     @staticmethod
171     def get_supported_algorithms() -> dict:
172         """Get information about supported PQC algorithms.
173
174         Returns:
175             dict: Supported algorithms with their properties
176         """
177         return {
178             "signature": {
179                 "ml-dsa-65": {
180                     "oqs_name": "Dilithium3",
181                     "public_key_size": 1952, # bytes
182                     "secret_key_size": 4000, # bytes
183                     "signature_size": 3293, # bytes
184                     "security_level": "NIST Level 3",
185                 },
186             },
187             "kem": {
188                 "ml-kem-768": {
189                     "oqs_name": "Kyber768",
190                     "public_key_size": 1184, # bytes
191                     "secret_key_size": 2400, # bytes
192                     "ciphertext_size": 1088, # bytes
193                     "shared_secret_size": 32, # bytes
194                     "security_level": "NIST Level 3",
195                 },
196             }
197         }
198
199     # Global instance
200     _liboqs_instance: Optional[LibOQSWrapper] = None
201
202
203     def get_liboqs() -> LibOQSWrapper:
204         """Get or create the global LibOQS wrapper instance.
205
206         Returns:
207             LibOQSWrapper: Global LibOQS wrapper instance
208         """
209         global _liboqs_instance
210         if _liboqs_instance is None:
211             _liboqs_instance = LibOQSWrapper()
212         return _liboqs_instance

```

Anhang 4.1.2: pqc_peer4_creator.py

Listing A-14

pqc_didpeer4_fm - pqc_peer4_creator.py

```

1 """Create did:peer:4 with ML-DSA-65 + ML-KEM-768."""
2
3 from typing import List, Optional, Sequence, Dict
4 from did_peer_4 import encode
5 from did_peer_4.input_doc import input_doc_from_keys_and_services, KeySpec
6
7 from acapy_agent.wallet.base import BaseWallet
8 from acapy_agent.wallet.did_info import DIDInfo
9 from acapy_agent.wallet.did_method import PEER4
10
11 from .key_types import ML_DSA_65, ML_KEM_768
12 from .pqc_multikey import key_info_to_multikey
13
14
15 async def create_pqc_peer4_did(
16     wallet: BaseWallet,
17     svc_endpoints: Optional[Sequence[str]] = None,
18     routing_keys: Optional[List[str]] = None,
19     metadata: Optional[Dict] = None,
20 ) -> DIDInfo:
21     """Create did:peer:4 with ML-DSA-65 (signature) + ML-KEM-768 (key agreement).
22
23     This function creates a PQC-enabled did:peer:4 DID with two keys:
24     1. ML-DSA-65 for authentication and assertion (digital signatures)
25     2. ML-KEM-768 for key agreement (encryption/key encapsulation)
26
27     Args:
28         wallet: Wallet instance for key management
29         svc_endpoints: Service endpoints for DIDComm messaging
30         routing_keys: Routing keys for mediation (optional)
31         metadata: Additional metadata to store with DID
32
33     Returns:
34         DIDInfo compatible with GET /wallet/did response format
35
36     Example:
37         >>> did_info = await create_pqc_peer4_did(
38             ...      wallet=wallet,
39             ...      svc_endpoints=["https://agent.example.com:8020"])
40             ...
41         >>> print(did_info.did)
42         did:peer:4:z6MNxxx...
43
44
45     # 1. Create ML-DSA-65 key (signature/authentication)
46     sig_key = await wallet.create_key(ML_DSA_65)
47
48     # 2. Create ML-KEM-768 key (key agreement/encryption)
49     kem_key = await wallet.create_key(ML_KEM_768)
50
51     # 3. Convert to multikeys

```

```

52     sig_multikey = key_info_to_multikey(sig_key)  # --> z6MN... (ML-DSA-65)
53     kem_multikey = key_info_to_multikey(kem_key)  # --> z6MK768... (ML-KEM-768)
54
55     # 4. Create KeySpec objects for did:peer:4
56     # NOTE: Order matters! did-peer-4 numbers from 0: key_specs[0] --> #key-0,
57     #       key_specs[1] --> #key-1
58     key_specs = [
59        KeySpec(
60             multikey=sig_multikey,
61             relationships=["authentication", "assertionMethod"] # --> #key-0
62         ),
63        KeySpec(
64             multikey=kem_multikey,
65             relationships=["keyAgreement"] # --> #key-1 (used in recipientKeys!)
66         ),
67     ]
68
69     # 5. Build DIDComm v1 services (compatible with DID Exchange 1.1)
70     services = []
71     for index, endpoint in enumerate(svc_endpoints or []):
72         services.append({
73             "id": f"#didcomm-{index}",
74             "type": "did-communication",
75             "recipientKeys": ["#key-1"], # ML-KEM-768 (key agreement/encryption,
76                                         key_specs[1] --> #key-1)
77             "routingKeys": routing_keys or [],
78             "serviceEndpoint": endpoint,
79             "priority": index,
80         })
81
82     # 6. Generate did:peer:4 (long form)
83     input_doc = input_doc_from_keys_and_services(
84         keys=key_specs,
85         services=services
86     )
87     did = encode(input_doc)
88
89     # 7. Create metadata (compatible with existing /wallet/did format)
90     did_metadata = metadata or {}
91     did_metadata.update({
92         "pqc_enabled": True,
93         "signature_algorithm": "ml-dsa-65",
94         "key_agreement_algorithm": "ml-kem-768",
95         "kem_key_kid": f"#{did}#key-1", # KEM key is key_specs[1] --> #key-1
96         "plugin": "pqc_didpeer4_fm",
97         "version": "0.1.0",
98     })
99
100    # 8. Create DIDInfo (SAME structure as original did:peer:4!)
101    did_info = DIDInfo(
102        did=did,
103        method=PEER4,
104        verkey=sig_key.verkey,
105        metadata=did_metadata,
106        key_type=ML_DSA_65,
107    )

```

```

107     # 9. Store DID in wallet
108     await wallet.store_did(did_info)
109
110     # 10. Assign Key IDs - did-peer-4 numbers from 0!
111     await wallet.assign_kid_to_key(sig_key.verkey, f"{did}#key-0") # key_specs[0] -->
112         #key-0
113     await wallet.assign_kid_to_key(kem_key.verkey, f"{did}#key-1") # key_specs[1] -->
114         #key-1
115
116     return did_info

```

Anhang 4.1.3: pqc_peer4_resolver.py

Listing A-15

pqc_didpeer4_fm - pqc_peer4_resolver.py

```

1 """PQC-aware did:peer:4 resolver."""
2
3 from re import compile
4 from typing import Optional, Pattern, Sequence, Text
5
6 from did_peer_4 import LONG_PATTERN, SHORT_PATTERN, long_to_short, resolve,
7     resolve_short
8
9 from acapy_agent.config.injection_context import InjectionContext
10 from acapy_agent.core.profile import Profile
11 from acapy_agent.resolver.base import BaseDIDResolver, DIDNotFound, ResolverType
12 from acapy_agent.storage.base import BaseStorage
13 from acapy_agent.storage.error import StorageNotFoundError
14 from acapy_agent.storage.record import StorageRecord
15
16
17 class PQCPeer4Resolver(BaseDIDResolver):
18     """Resolver for PQC-enabled did:peer:4 DIDs.
19
20     Uses external did-peer-4 library for resolution.
21     PQC multikeys (ML-DSA-65, ML-KEM-768) are preserved in verification methods.
22
23     This resolver is registered automatically by the pqc_didpeer4_fm plugin
24     and handles all did:peer:4 DIDs, whether they use PQC or classical algorithms.
25     """
26
27     RECORD_TYPE = "long_peer_did_4_doc"
28
29     def __init__(self):
30         """Initialize PQC Peer4 Resolver."""
31         super().__init__(ResolverType.NATIVE)
32
33     async def setup(self, context: InjectionContext):
34         """Setup resolver (no initialization needed)."""
35
36     @property
37     def supported_did_regex(self) -> Pattern:

```

```

37     """Return supported_did_regex for did:peer:4."""
38     return compile(f"{LONG_PATTERN.pattern}|{SHORT_PATTERN.pattern}")
39
40     async def _resolve(
41         self,
42         profile: Profile,
43         did: str,
44         service_accept: Optional[Sequence[Text]] = None,
45     ) -> dict:
46         """Resolve did:peer:4 DID document.
47
48     Args:
49         profile: Profile context
50         did: The did:peer:4 DID to resolve
51         service_accept: Service types to accept (optional)
52
53     Returns:
54         DID Document as dict
55
56     Raises:
57         DIDNotFound: If DID resolution fails
58     """
59     if LONG_PATTERN.match(did):
60         short_did_peer_4 = long_to_short(did)
61         # resolve and save long form
62         async with profile.session() as session:
63             storage = session.inject(BaseStorage)
64             try:
65                 record = await storage.get_record(self.RECORD_TYPE,
66                                                 short_did_peer_4)
67             except StorageNotFoundError:
68                 record = StorageRecord(self.RECORD_TYPE, did, {}, short_did_peer_4)
69                 await storage.add_record(record)
70             document = resolve(did)
71
72     elif SHORT_PATTERN.match(did):
73         async with profile.session() as session:
74             storage = session.inject(BaseStorage)
75             try:
76                 record = await storage.get_record(self.RECORD_TYPE, did)
77             except StorageNotFoundError:
78                 raise DIDNotFound(
79                     f"short did:peer:4 does not correspond to a known long did:
80                         peer:4: {did}"
81                 )
82             document = resolve_short(record.value)
83     else:
84         raise ValueError(f"{did} did not match long or short form of did:peer:4")
85
86     return document

```

Anhang 4.1.4: pqc_multicodec.py

Listing A-16

pqc_didpeer4_fm - pqc_multicodec.py

```

1 """PQC Multicodec registry for ML-DSA and ML-KEM algorithms.
2
3 This module provides a standalone multicodec registry for PQC algorithms,
4 independent of ACA-Py's built-in multicodec system which uses a fixed Enum.
5 """
6
7 # PQC Multicodec prefixes (provisional - based on W3C draft)
8 # https://w3c-ccg.github.io/multicodec/
9 PQC_MULTICODECS = {
10     # ML-DSA (NIST FIPS-204) - Digital Signature Algorithm
11     "ml-dsa-44-pub": b"\xd0\x44",
12     "ml-dsa-65-pub": b"\xd0\x65", # Primary signature algorithm
13     "ml-dsa-87-pub": b"\xd0\x87",
14
15     # ML-KEM (NIST FIPS-203) - Key Encapsulation Mechanism
16     "ml-kem-512-pub": b"\xe0\x12",
17     "ml-kem-768-pub": b"\xe0\x18", # Primary key agreement algorithm
18     "ml-kem-1024-pub": b"\xe0\x24",
19 }
20
21
22 def register_pqc_multicodecs():
23     """Register PQC multicodecs.
24
25     This is a no-op since the registry is already defined as a module-level
26     dictionary. The function exists for API compatibility with the plugin setup.
27     """
28     pass
29
30
31 def wrap_pqc(codec_name: str, data: bytes) -> bytes:
32     """Wrap data with PQC multicodec prefix.
33
34     Args:
35         codec_name: Multicodec name (e.g., "ml-dsa-65-pub")
36         data: Raw key bytes to wrap
37
38     Returns:
39         Multicodec-prefixed bytes
40
41     Raises:
42         ValueError: If codec_name is not a known PQC codec
43     """
44     if codec_name not in PQC_MULTICODECS:
45         raise ValueError(
46             f"Unknown PQC codec: {codec_name}. "
47             f"Supported: {list(PQC_MULTICODECS.keys())}"
48         )
49     return PQC_MULTICODECS[codec_name] + data
50
51

```

```

52| def unwrap_pqc(data: bytes) -> tuple[str, bytes]:
53|     """Unwrap PQC multicodec prefix from data.
54|
55|     Args:
56|         data: Multicodec-prefixed key bytes
57|
58|     Returns:
59|         Tuple of (codec_name, raw_key_bytes)
60|
61|     Raises:
62|         ValueError: If data doesn't start with a known PQC multicodec prefix
63|     """
64|     for codec_name, prefix in PQC_MULTICODECS.items():
65|         if data.startswith(prefix):
66|             return codec_name, data[len(prefix):]
67|
68|     raise ValueError(
69|         f"Unknown PQC multicodec prefix. "
70|         f"Data starts with: {data[:2].hex() if len(data) >= 2 else 'empty'}"
71|     )

```

Anhang 4.1.5: pqc_multidekey.py

Listing A-17

pqc_didpeer4_fm - pqc_multidekey.py

```

1  """Convert PQC KeyInfo to Multikey format for did:peer:4."""
2
3  from base58 import b58decode
4  from multiformats import multibase
5  from acapy_agent.wallet.did_info import KeyInfo
6
7  from .key_types import ML_DSA_65, ML_KEM_768
8  from .pqc_multicodec import wrap_pqc, unwrap_pqc
9
10
11 # Mapping: KeyType --> Multicodec name
12 KEY_TYPE_TO_MULTICODEC = {
13     ML_DSA_65.key_type: "ml-dsa-65-pub",
14     ML_KEM_768.key_type: "ml-kem-768-pub",
15 }
16
17
18 def key_info_to_multikey(key_info: KeyInfo) -> str:
19     """Convert PQC KeyInfo to multikey string.
20
21     Args:
22         key_info: KeyInfo with ML-DSA-65 or ML-KEM-768 key
23
24     Returns:
25         Multikey string (e.g., "z6MNx8r2..." for ML-DSA-65)
26
27     Raises:

```

```

28         ValueError: If key type is not supported
29
30     Example:
31         >>> key = await wallet.create_key(ML_DSA_65)
32         >>> multikey = key_info_to_multic平ey(key)
33         >>> print(multikey)
34         z6MNxxx... # ML-DSC-65 multic平ey (base58btc)
35
36     # Get multicodec name for this key type
37     codec_name = KEY_TYPE_TO_MULTICODEC.get(key_info.key_type.key_type)
38     if not codec_name:
39         raise ValueError(
40             f"Unsupported key type for PQC multic平ey: {key_info.key_type.key_type}. "
41             f"Supported: {list(KEY_TYPE_TO_MULTICODEC.keys())}"
42         )
43
44     # Decode base58 verkey --> raw bytes
45     raw_key = b58decode(key_info.verkey)
46
47     # Wrap with PQC multicodec prefix
48     multicodec_key = wrap_pqc(codec_name, raw_key)
49
50     # Encode with multibase (base58btc --> starts with 'z')
51     multikey = multibase.encode(multic平ey_key, "base58btc")
52
53     return multikey
54
55
56 def multikey_to_raw(multikey: str) -> tuple:
57     """Decode multikey to (codec_name, raw_bytes).
58
59     Used for verification and key recovery.
60
61     Args:
62         multikey: Multikey string (e.g., "z6MNxxx...")
```

63

64 Returns:

65 Tuple of (codec_name, raw_key_bytes)

66

67 Example:

68 >>> codec, raw = multikey_to_raw("z6MNxxx...")
69 >>> print(codec)
70 ml-dsa-65-pub
71
72 # Decode multibase
73 decoded = multibase.decode(multikey)
74
75 # Unwrap PQC multicodec
76 codec_name, raw_key = unwrap_pqc(decoded)
77
78 return codec_name, raw_key

Anhang 4.1.6: pqc_didcomm_v1.py

Listing A-18

pqc_didpeer4_fm - pqc_didcomm_v1.py

```

1 """PQC-aware DIDComm v1 envelope handling.
2
3 This module provides Post-Quantum Cryptography (PQC) support for DIDComm v1
4 pack/unpack operations while maintaining compatibility with classical ED25519/X25519.
5
6 Classical DIDComm v1:
7     - Uses ED25519 (signatures) --> X25519 (ECDH) conversion
8     - crypto_box (ECDH + XChaCha20-Poly1305) for key exchange
9     - Not quantum-safe!
10
11 PQC-DIDComm v1:
12     - Uses ML-DSA-65 (signatures) for authcrypt
13     - ML-KEM-768 (KEM) for CEK encapsulation (replaces ECDH)
14     - XChaCha20-Poly1305 for symmetric encryption (quantum-safe!)
15     - JWE format: alg = "PQC-Authcrypt" or "PQC-Anoncrypt"
16
17 This implementation supports:
18     - Pure PQC mode (ML-DSA-65 + ML-KEM-768)
19     - Pure classical mode (ED25519 + X25519)
20     - Hybrid mode (mixed PQC/classical recipients)
21 """
22
23 import logging
24 import json
25 from collections import OrderedDict
26 from typing import Optional, Sequence, Tuple, Union
27
28 from aries_askar import Key, KeyAlg, Session
29 from aries_askar.bindings import key_get_secret_bytes
30
31 from .liboqs_wrapper import get_liboqs
32 from .key_types import ML_DSA_65, ML_KEM_768
33 from .askar_pqc_patch import PQCKey
34
35 LOGGER = logging.getLogger(__name__)
36
37
38 def _is_pqc_key(key: Union[Key, PQCKey]) -> bool:
39     """Check if key is a PQC key (ML-DSA-65 or ML-KEM-768).
40
41     Args:
42         key: Key to check
43
44     Returns:
45         bool: True if PQC key, False if classical key
46     """
47     if isinstance(key, PQCKey):
48         return True
49     return False
50
51

```

```

52| def _detect_recipient_key_type(verkey: str) -> str:
53|     """Detect if a base58 verkey is PQC or classical based on length.
54|
55|     ML-KEM-768 public keys: 1184 bytes --> ~1615 base58 chars
56|     ED25519 public keys: 32 bytes --> 44 base58 chars
57|
58|     Args:
59|         verkey: Base58-encoded public key
60|
61|     Returns:
62|         str: "pqc" or "classical"
63|     """
64|     if len(verkey) > 1000:  # Definitely PQC
65|         return "pqc"
66|     else:
67|         return "classical"
68|
69|
70| async def pack_message_pqc(
71|     session: Session,
72|     to_verkeys: Sequence[str],
73|     from_key: Optional[Union[Key, PQCKey]],
74|     message: str,
75| ) -> bytes:
76|     """Pack a message with PQC support.
77|
78|     This function automatically detects whether to use PQC or classical
79|     crypto based on the sender key type and recipient key types.
80|
81|     Args:
82|         session: Askar session for key lookups
83|         to_verkeys: List of recipient verkeys (base58)
84|         from_key: Sender key (PQCKey or aries_askar.Key) or None for anoncrypt
85|         message: Message to pack (string or bytes)
86|
87|     Returns:
88|         bytes: JWE-encoded packed message
89|     """
90|
91|     # Detect crypto mode
92|     sender_is_pqc = _is_pqc_key(from_key) if from_key else False
93|     recipient_types = [_detect_recipient_key_type(vk) for vk in to_verkeys]
94|
95|     # Check if any recipient is PQC
96|     any_pqc = sender_is_pqc or "pqc" in recipient_types
97|     all_pqc = sender_is_pqc and all(t == "pqc" for t in recipient_types)
98|
99|     if any_pqc:
100|         LOGGER.info(f"Using PQC pack mode: sender_pqc={sender_is_pqc}, "
101|                     f"recipients={recipient_types}")
102|         return await _pack_pqc(session, to_verkeys, from_key, message)
103|     else:
104|         # Delegate to classical implementation
105|         LOGGER.debug("Using classical pack mode (ED25519/X25519)")
106|         from acapy_agent.askar.didcomm.v1 import pack_message as pack_classical
107|
108|         # Convert message to bytes if needed

```

```

108     message_bytes = message.encode("utf-8") if isinstance(message, str) else
109         message
110
111     return pack_classical(to_verkeys, from_key, message_bytes)
112
113     async def _pack_pqc(
114         session: Session,
115         to_verkeys: Sequence[str],
116         from_key: Optional[Union[Key, PQCKey]],
117         message: Union[str, bytes],
118     ) -> bytes:
119         """Pack a message using PQC algorithms (ML-KEM-768 + XChaCha20-Poly1305).
120
121         Algorithm:
122             1. Generate random CEK (Content Encryption Key) using ChaCha20-Poly1305
123             2. For each recipient:
124                 - Fetch ML-KEM-768 public key
125                 - Encapsulate CEK using ML-KEM-768
126                 - Add JWE recipient with encrypted_key
127             3. If authcrypt:
128                 - Sign CEK with ML-DSA-65
129                 - Add signature to protected header
130             4. Encrypt message with CEK using ChaCha20-Poly1305
131             5. Return JWE envelope
132
133         Args:
134             session: Askar session for key lookups
135             to_verkeys: Recipient verkeys (base58)
136             from_key: Sender key (PQCKey with ML-DSA-65 or ML-KEM-768) or None
137             message: Message to encrypt
138
139         Returns:
140             bytes: JWE-encoded message
141         """
142
143         from acapy_agent.utils.jwe import JweEnvelope, JweRecipient, b64url
144         from acapy_agent.wallet.util import bytes_to_b58, b58_to_bytes
145
146         # Convert message to bytes
147         message_bytes = message.encode("utf-8") if isinstance(message, str) else message
148
149         # 1. Generate CEK (Content Encryption Key)
150         cek = Key.generate(KeyAlg.C20P) # ChaCha20-Poly1305
151         cek_bytes = key_get_secret_bytes(cek._handle)
152
153         # Get liboqs for KEM operations
154         liboqs = get_liboqs()
155
156         # 2. Prepare sender info for authcrypt
157         sender_vk = None
158         if from_key:
159             if isinstance(from_key, PQCKey):
160                 sender_vk = bytes_to_b58(from_key.get_public_bytes()).encode("utf-8")
161             else:
162                 sender_vk = bytes_to_b58(from_key.get_public_bytes()).encode("utf-8")
163
164         # 3. Create JWE wrapper

```

```

163     wrapper = JweEnvelope(with_protected_recipients=True, with_flatten_recipients=False)
164
165     # 4. Encapsulate CEK for each recipient
166     for target_vk in to_verkeys:
167         # Decode recipient's public key from base58
168         # Note: Unlike sender keys, recipient keys are NOT stored in our wallet!
169         # The verkey is passed as a base58-encoded public key that we decode directly.
170
171         # Detect key type based on length (same as pack_message_pqc)
172         recipient_type = _detect_recipient_key_type(target_vk)
173
174         if recipient_type == "pqc":
175             # PQC path: Decode ML-KEM-768 public key from base58
176             LOGGER.debug(f"Decoding ML-KEM-768 recipient key: {target_vk[:20]}...")
177             target_public = b58_to_bytes(target_vk)
178
179         if sender_vk:
180             # Authcrypt mode: Encapsulate CEK + sign with sender
181             ciphertext, shared_secret = liboqs.kem_encapsulate(target_public)
182
183             # For authcrypt, we need to sign the ciphertext with sender's ML-DSA
184             -65
185             if isinstance(from_key, PQCKey) and from_key.algorithm == "ml-dsa-65":
186                 signature = liboqs.ml_dsa_sign(
187                     from_key.get_secret_bytes(),
188                     ciphertext
189                 )
190
191             wrapper.add_recipient(
192                 JweRecipient(
193                     encrypted_key=ciphertext,
194                     header=OrderedDict([
195                         ("kid", target_vk),
196                         ("sender", b64url(sender_vk)),
197                         ("sig", b64url(signature)),
198                     ])
199                 )
200             )
201         else:
202             # Sender is not ML-DSA-65, can't do PQC authcrypt
203             raise ValueError("Authcrypt requires sender with ML-DSA-65 key")
204
205             # Use shared_secret as CEK (for simplicity, in production use KDF)
206             # Note: This is a simplified implementation!
207             cek_bytes = shared_secret[:32] # ChaCha20 needs 256-bit key
208             cek = Key.from_secret_bytes(KeyAlg.C20P, cek_bytes) # Recreate CEK
209             from shared_secret!
210
211         else:
212             # Anoncrypt mode: Just encapsulate CEK
213             ciphertext, shared_secret = liboqs.kem_encapsulate(target_public)
214
215             wrapper.add_recipient(
216                 JweRecipient(
217                     encrypted_key=ciphertext,
218                     header={"kid": target_vk}
219                 )
220             )
221
222         if len(wrapper.recipients) > 1:
223             raise ValueError("JweEnvelope can only have one recipient")
224
225         return wrapper
226
227     def __init__(self, recipients, header=None):
228         self.recipients = recipients
229         self.header = header
230
231     def __str__(self):
232         return f"JweEnvelope(recipients={self.recipients}, header={self.header})"
233
234     def __repr__(self):
235         return str(self)
236
237     def __eq__(self, other):
238         if not isinstance(other, JweEnvelope):
239             return False
240
241         if self.recipients != other.recipients:
242             return False
243
244         if self.header != other.header:
245             return False
246
247         return True
248
249     def __ne__(self, other):
250         return not self == other
251
252     def __hash__(self):
253         return hash((self.recipients, self.header))
254
255     def __len__(self):
256         return len(self.recipients)
257
258     def __iter__(self):
259         return iter(self.recipients)
260
261     def __contains__(self, item):
262         return item in self.recipients
263
264     def __getitem__(self, index):
265         return self.recipients[index]
266
267     def __setitem__(self, index, value):
268         self.recipients[index] = value
269
270     def __delitem__(self, index):
271         del self.recipients[index]
272
273     def __add__(self, other):
274         if not isinstance(other, JweEnvelope):
275             raise TypeError("Can only add another JweEnvelope object")
276
277         new_recipients = self.recipients + other.recipients
278
279         new_header = self.header
280         if other.header:
281             new_header = {**self.header, **other.header}
282
283         return JweEnvelope(new_recipients, new_header)
284
285     def __iadd__(self, other):
286         if not isinstance(other, JweEnvelope):
287             raise TypeError("Can only add another JweEnvelope object")
288
289         self.recipients += other.recipients
290
291         if other.header:
292             self.header = {**self.header, **other.header}
293
294         return self
295
296     def __radd__(self, other):
297         if not isinstance(other, JweEnvelope):
298             raise TypeError("Can only add another JweEnvelope object")
299
300         new_recipients = other.recipients + self.recipients
301
302         new_header = other.header
303         if self.header:
304             new_header = {**other.header, **self.header}
305
306         return JweEnvelope(new_recipients, new_header)
307
308     def __rradd__(self, other):
309         if not isinstance(other, JweEnvelope):
310             raise TypeError("Can only add another JweEnvelope object")
311
312         self.recipients += other.recipients
313
314         if other.header:
315             self.header = {**other.header, **self.header}
316
317         return self
318
319     def __sub__(self, other):
320         if not isinstance(other, JweEnvelope):
321             raise TypeError("Can only subtract another JweEnvelope object")
322
323         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
324
325         new_header = self.header
326         if other.header:
327             new_header = {**self.header, **other.header}
328
329         return JweEnvelope(new_recipients, new_header)
330
331     def __isub__(self, other):
332         if not isinstance(other, JweEnvelope):
333             raise TypeError("Can only subtract another JweEnvelope object")
334
335         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
336
337         if other.header:
338             self.header = {**other.header, **self.header}
339
340         return self
341
342     def __eq__(self, other):
343         if not isinstance(other, JweEnvelope):
344             return False
345
346         if self.recipients != other.recipients:
347             return False
348
349         if self.header != other.header:
350             return False
351
352         return True
353
354     def __ne__(self, other):
355         return not self == other
356
357     def __hash__(self):
358         return hash((self.recipients, self.header))
359
360     def __len__(self):
361         return len(self.recipients)
362
363     def __iter__(self):
364         return iter(self.recipients)
365
366     def __contains__(self, item):
367         return item in self.recipients
368
369     def __getitem__(self, index):
370         return self.recipients[index]
371
372     def __setitem__(self, index, value):
373         self.recipients[index] = value
374
375     def __delitem__(self, index):
376         del self.recipients[index]
377
378     def __add__(self, other):
379         if not isinstance(other, JweEnvelope):
380             raise TypeError("Can only add another JweEnvelope object")
381
382         new_recipients = self.recipients + other.recipients
383
384         new_header = self.header
385         if other.header:
386             new_header = {**self.header, **other.header}
387
388         return JweEnvelope(new_recipients, new_header)
389
390     def __iadd__(self, other):
391         if not isinstance(other, JweEnvelope):
392             raise TypeError("Can only add another JweEnvelope object")
393
394         self.recipients += other.recipients
395
396         if other.header:
397             self.header = {**other.header, **self.header}
398
399         return self
400
401     def __radd__(self, other):
402         if not isinstance(other, JweEnvelope):
403             raise TypeError("Can only add another JweEnvelope object")
404
405         new_recipients = other.recipients + self.recipients
406
407         new_header = other.header
408         if self.header:
409             new_header = {**other.header, **self.header}
410
411         return JweEnvelope(new_recipients, new_header)
412
413     def __rradd__(self, other):
414         if not isinstance(other, JweEnvelope):
415             raise TypeError("Can only add another JweEnvelope object")
416
417         self.recipients += other.recipients
418
419         if other.header:
420             self.header = {**other.header, **self.header}
421
422         return self
423
424     def __sub__(self, other):
425         if not isinstance(other, JweEnvelope):
426             raise TypeError("Can only subtract another JweEnvelope object")
427
428         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
429
430         new_header = self.header
431         if other.header:
432             new_header = {**self.header, **other.header}
433
434         return JweEnvelope(new_recipients, new_header)
435
436     def __isub__(self, other):
437         if not isinstance(other, JweEnvelope):
438             raise TypeError("Can only subtract another JweEnvelope object")
439
440         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
441
442         if other.header:
443             self.header = {**other.header, **self.header}
444
445         return self
446
447     def __eq__(self, other):
448         if not isinstance(other, JweEnvelope):
449             return False
450
451         if self.recipients != other.recipients:
452             return False
453
454         if self.header != other.header:
455             return False
456
457         return True
458
459     def __ne__(self, other):
460         return not self == other
461
462     def __hash__(self):
463         return hash((self.recipients, self.header))
464
465     def __len__(self):
466         return len(self.recipients)
467
468     def __iter__(self):
469         return iter(self.recipients)
470
471     def __contains__(self, item):
472         return item in self.recipients
473
474     def __getitem__(self, index):
475         return self.recipients[index]
476
477     def __setitem__(self, index, value):
478         self.recipients[index] = value
479
480     def __delitem__(self, index):
481         del self.recipients[index]
482
483     def __add__(self, other):
484         if not isinstance(other, JweEnvelope):
485             raise TypeError("Can only add another JweEnvelope object")
486
487         new_recipients = self.recipients + other.recipients
488
489         new_header = self.header
490         if other.header:
491             new_header = {**self.header, **other.header}
492
493         return JweEnvelope(new_recipients, new_header)
494
495     def __iadd__(self, other):
496         if not isinstance(other, JweEnvelope):
497             raise TypeError("Can only add another JweEnvelope object")
498
499         self.recipients += other.recipients
500
501         if other.header:
502             self.header = {**other.header, **self.header}
503
504         return self
505
506     def __radd__(self, other):
507         if not isinstance(other, JweEnvelope):
508             raise TypeError("Can only add another JweEnvelope object")
509
510         new_recipients = other.recipients + self.recipients
511
512         new_header = other.header
513         if self.header:
514             new_header = {**other.header, **self.header}
515
516         return JweEnvelope(new_recipients, new_header)
517
518     def __rradd__(self, other):
519         if not isinstance(other, JweEnvelope):
520             raise TypeError("Can only add another JweEnvelope object")
521
522         self.recipients += other.recipients
523
524         if other.header:
525             self.header = {**other.header, **self.header}
526
527         return self
528
529     def __sub__(self, other):
530         if not isinstance(other, JweEnvelope):
531             raise TypeError("Can only subtract another JweEnvelope object")
532
533         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
534
535         new_header = self.header
536         if other.header:
537             new_header = {**self.header, **other.header}
538
539         return JweEnvelope(new_recipients, new_header)
540
541     def __isub__(self, other):
542         if not isinstance(other, JweEnvelope):
543             raise TypeError("Can only subtract another JweEnvelope object")
544
545         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
546
547         if other.header:
548             self.header = {**other.header, **self.header}
549
550         return self
551
552     def __eq__(self, other):
553         if not isinstance(other, JweEnvelope):
554             return False
555
556         if self.recipients != other.recipients:
557             return False
558
559         if self.header != other.header:
560             return False
561
562         return True
563
564     def __ne__(self, other):
565         return not self == other
566
567     def __hash__(self):
568         return hash((self.recipients, self.header))
569
570     def __len__(self):
571         return len(self.recipients)
572
573     def __iter__(self):
574         return iter(self.recipients)
575
576     def __contains__(self, item):
577         return item in self.recipients
578
579     def __getitem__(self, index):
580         return self.recipients[index]
581
582     def __setitem__(self, index, value):
583         self.recipients[index] = value
584
585     def __delitem__(self, index):
586         del self.recipients[index]
587
588     def __add__(self, other):
589         if not isinstance(other, JweEnvelope):
590             raise TypeError("Can only add another JweEnvelope object")
591
592         new_recipients = self.recipients + other.recipients
593
594         new_header = self.header
595         if other.header:
596             new_header = {**self.header, **other.header}
597
598         return JweEnvelope(new_recipients, new_header)
599
600     def __iadd__(self, other):
601         if not isinstance(other, JweEnvelope):
602             raise TypeError("Can only add another JweEnvelope object")
603
604         self.recipients += other.recipients
605
606         if other.header:
607             self.header = {**other.header, **self.header}
608
609         return self
610
611     def __radd__(self, other):
612         if not isinstance(other, JweEnvelope):
613             raise TypeError("Can only add another JweEnvelope object")
614
615         new_recipients = other.recipients + self.recipients
616
617         new_header = other.header
618         if self.header:
619             new_header = {**other.header, **self.header}
620
621         return JweEnvelope(new_recipients, new_header)
622
623     def __rradd__(self, other):
624         if not isinstance(other, JweEnvelope):
625             raise TypeError("Can only add another JweEnvelope object")
626
627         self.recipients += other.recipients
628
629         if other.header:
630             self.header = {**other.header, **self.header}
631
632         return self
633
634     def __sub__(self, other):
635         if not isinstance(other, JweEnvelope):
636             raise TypeError("Can only subtract another JweEnvelope object")
637
638         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
639
640         new_header = self.header
641         if other.header:
642             new_header = {**self.header, **other.header}
643
644         return JweEnvelope(new_recipients, new_header)
645
646     def __isub__(self, other):
647         if not isinstance(other, JweEnvelope):
648             raise TypeError("Can only subtract another JweEnvelope object")
649
650         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
651
652         if other.header:
653             self.header = {**other.header, **self.header}
654
655         return self
656
657     def __eq__(self, other):
658         if not isinstance(other, JweEnvelope):
659             return False
660
661         if self.recipients != other.recipients:
662             return False
663
664         if self.header != other.header:
665             return False
666
667         return True
668
669     def __ne__(self, other):
670         return not self == other
671
672     def __hash__(self):
673         return hash((self.recipients, self.header))
674
675     def __len__(self):
676         return len(self.recipients)
677
678     def __iter__(self):
679         return iter(self.recipients)
680
681     def __contains__(self, item):
682         return item in self.recipients
683
684     def __getitem__(self, index):
685         return self.recipients[index]
686
687     def __setitem__(self, index, value):
688         self.recipients[index] = value
689
690     def __delitem__(self, index):
691         del self.recipients[index]
692
693     def __add__(self, other):
694         if not isinstance(other, JweEnvelope):
695             raise TypeError("Can only add another JweEnvelope object")
696
697         new_recipients = self.recipients + other.recipients
698
699         new_header = self.header
700         if other.header:
701             new_header = {**self.header, **other.header}
702
703         return JweEnvelope(new_recipients, new_header)
704
705     def __iadd__(self, other):
706         if not isinstance(other, JweEnvelope):
707             raise TypeError("Can only add another JweEnvelope object")
708
709         self.recipients += other.recipients
710
711         if other.header:
712             self.header = {**other.header, **self.header}
713
714         return self
715
716     def __radd__(self, other):
717         if not isinstance(other, JweEnvelope):
718             raise TypeError("Can only add another JweEnvelope object")
719
720         new_recipients = other.recipients + self.recipients
721
722         new_header = other.header
723         if self.header:
724             new_header = {**other.header, **self.header}
725
726         return JweEnvelope(new_recipients, new_header)
727
728     def __rradd__(self, other):
729         if not isinstance(other, JweEnvelope):
730             raise TypeError("Can only add another JweEnvelope object")
731
732         self.recipients += other.recipients
733
734         if other.header:
735             self.header = {**other.header, **self.header}
736
737         return self
738
739     def __sub__(self, other):
740         if not isinstance(other, JweEnvelope):
741             raise TypeError("Can only subtract another JweEnvelope object")
742
743         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
744
745         new_header = self.header
746         if other.header:
747             new_header = {**self.header, **other.header}
748
749         return JweEnvelope(new_recipients, new_header)
750
751     def __isub__(self, other):
752         if not isinstance(other, JweEnvelope):
753             raise TypeError("Can only subtract another JweEnvelope object")
754
755         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
756
757         if other.header:
758             self.header = {**other.header, **self.header}
759
760         return self
761
762     def __eq__(self, other):
763         if not isinstance(other, JweEnvelope):
764             return False
765
766         if self.recipients != other.recipients:
767             return False
768
769         if self.header != other.header:
770             return False
771
772         return True
773
774     def __ne__(self, other):
775         return not self == other
776
777     def __hash__(self):
778         return hash((self.recipients, self.header))
779
780     def __len__(self):
781         return len(self.recipients)
782
783     def __iter__(self):
784         return iter(self.recipients)
785
786     def __contains__(self, item):
787         return item in self.recipients
788
789     def __getitem__(self, index):
790         return self.recipients[index]
791
792     def __setitem__(self, index, value):
793         self.recipients[index] = value
794
795     def __delitem__(self, index):
796         del self.recipients[index]
797
798     def __add__(self, other):
799         if not isinstance(other, JweEnvelope):
800             raise TypeError("Can only add another JweEnvelope object")
801
802         new_recipients = self.recipients + other.recipients
803
804         new_header = self.header
805         if other.header:
806             new_header = {**self.header, **other.header}
807
808         return JweEnvelope(new_recipients, new_header)
809
810     def __iadd__(self, other):
811         if not isinstance(other, JweEnvelope):
812             raise TypeError("Can only add another JweEnvelope object")
813
814         self.recipients += other.recipients
815
816         if other.header:
817             self.header = {**other.header, **self.header}
818
819         return self
820
821     def __radd__(self, other):
822         if not isinstance(other, JweEnvelope):
823             raise TypeError("Can only add another JweEnvelope object")
824
825         new_recipients = other.recipients + self.recipients
826
827         new_header = other.header
828         if self.header:
829             new_header = {**other.header, **self.header}
830
831         return JweEnvelope(new_recipients, new_header)
832
833     def __rradd__(self, other):
834         if not isinstance(other, JweEnvelope):
835             raise TypeError("Can only add another JweEnvelope object")
836
837         self.recipients += other.recipients
838
839         if other.header:
840             self.header = {**other.header, **self.header}
841
842         return self
843
844     def __sub__(self, other):
845         if not isinstance(other, JweEnvelope):
846             raise TypeError("Can only subtract another JweEnvelope object")
847
848         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
849
850         new_header = self.header
851         if other.header:
852             new_header = {**self.header, **other.header}
853
854         return JweEnvelope(new_recipients, new_header)
855
856     def __isub__(self, other):
857         if not isinstance(other, JweEnvelope):
858             raise TypeError("Can only subtract another JweEnvelope object")
859
860         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
861
862         if other.header:
863             self.header = {**other.header, **self.header}
864
865         return self
866
867     def __eq__(self, other):
868         if not isinstance(other, JweEnvelope):
869             return False
870
871         if self.recipients != other.recipients:
872             return False
873
874         if self.header != other.header:
875             return False
876
877         return True
878
879     def __ne__(self, other):
880         return not self == other
881
882     def __hash__(self):
883         return hash((self.recipients, self.header))
884
885     def __len__(self):
886         return len(self.recipients)
887
888     def __iter__(self):
889         return iter(self.recipients)
890
891     def __contains__(self, item):
892         return item in self.recipients
893
894     def __getitem__(self, index):
895         return self.recipients[index]
896
897     def __setitem__(self, index, value):
898         self.recipients[index] = value
899
900     def __delitem__(self, index):
901         del self.recipients[index]
902
903     def __add__(self, other):
904         if not isinstance(other, JweEnvelope):
905             raise TypeError("Can only add another JweEnvelope object")
906
907         new_recipients = self.recipients + other.recipients
908
909         new_header = self.header
910         if other.header:
911             new_header = {**self.header, **other.header}
912
913         return JweEnvelope(new_recipients, new_header)
914
915     def __iadd__(self, other):
916         if not isinstance(other, JweEnvelope):
917             raise TypeError("Can only add another JweEnvelope object")
918
919         self.recipients += other.recipients
920
921         if other.header:
922             self.header = {**other.header, **self.header}
923
924         return self
925
926     def __radd__(self, other):
927         if not isinstance(other, JweEnvelope):
928             raise TypeError("Can only add another JweEnvelope object")
929
930         new_recipients = other.recipients + self.recipients
931
932         new_header = other.header
933         if self.header:
934             new_header = {**other.header, **self.header}
935
936         return JweEnvelope(new_recipients, new_header)
937
938     def __rradd__(self, other):
939         if not isinstance(other, JweEnvelope):
940             raise TypeError("Can only add another JweEnvelope object")
941
942         self.recipients += other.recipients
943
944         if other.header:
945             self.header = {**other.header, **self.header}
946
947         return self
948
949     def __sub__(self, other):
950         if not isinstance(other, JweEnvelope):
951             raise TypeError("Can only subtract another JweEnvelope object")
952
953         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
954
955         new_header = self.header
956         if other.header:
957             new_header = {**self.header, **other.header}
958
959         return JweEnvelope(new_recipients, new_header)
960
961     def __isub__(self, other):
962         if not isinstance(other, JweEnvelope):
963             raise TypeError("Can only subtract another JweEnvelope object")
964
965         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
966
967         if other.header:
968             self.header = {**other.header, **self.header}
969
970         return self
971
972     def __eq__(self, other):
973         if not isinstance(other, JweEnvelope):
974             return False
975
976         if self.recipients != other.recipients:
977             return False
978
979         if self.header != other.header:
980             return False
981
982         return True
983
984     def __ne__(self, other):
985         return not self == other
986
987     def __hash__(self):
988         return hash((self.recipients, self.header))
989
990     def __len__(self):
991         return len(self.recipients)
992
993     def __iter__(self):
994         return iter(self.recipients)
995
996     def __contains__(self, item):
997         return item in self.recipients
998
999     def __getitem__(self, index):
1000        return self.recipients[index]
1001
1002     def __setitem__(self, index, value):
1003        self.recipients[index] = value
1004
1005     def __delitem__(self, index):
1006        del self.recipients[index]
1007
1008     def __add__(self, other):
1009        if not isinstance(other, JweEnvelope):
1010            raise TypeError("Can only add another JweEnvelope object")
1011
1012        new_recipients = self.recipients + other.recipients
1013
1014        new_header = self.header
1015        if other.header:
1016            new_header = {**self.header, **other.header}
1017
1018        return JweEnvelope(new_recipients, new_header)
1019
1020     def __iadd__(self, other):
1021         if not isinstance(other, JweEnvelope):
1022             raise TypeError("Can only add another JweEnvelope object")
1023
1024         self.recipients += other.recipients
1025
1026         if other.header:
1027             self.header = {**other.header, **self.header}
1028
1029         return self
1030
1031     def __radd__(self, other):
1032         if not isinstance(other, JweEnvelope):
1033             raise TypeError("Can only add another JweEnvelope object")
1034
1035         new_recipients = other.recipients + self.recipients
1036
1037         new_header = other.header
1038         if self.header:
1039             new_header = {**other.header, **self.header}
1040
1041         return JweEnvelope(new_recipients, new_header)
1042
1043     def __rradd__(self, other):
1044         if not isinstance(other, JweEnvelope):
1045             raise TypeError("Can only add another JweEnvelope object")
1046
1047         self.recipients += other.recipients
1048
1049         if other.header:
1050             self.header = {**other.header, **self.header}
1051
1052         return self
1053
1054     def __sub__(self, other):
1055         if not isinstance(other, JweEnvelope):
1056             raise TypeError("Can only subtract another JweEnvelope object")
1057
1058         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1059
1060         new_header = self.header
1061         if other.header:
1062             new_header = {**self.header, **other.header}
1063
1064         return JweEnvelope(new_recipients, new_header)
1065
1066     def __isub__(self, other):
1067         if not isinstance(other, JweEnvelope):
1068             raise TypeError("Can only subtract another JweEnvelope object")
1069
1070         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1071
1072         if other.header:
1073             self.header = {**other.header, **self.header}
1074
1075         return self
1076
1077     def __eq__(self, other):
1078         if not isinstance(other, JweEnvelope):
1079             return False
1080
1081         if self.recipients != other.recipients:
1082             return False
1083
1084         if self.header != other.header:
1085             return False
1086
1087         return True
1088
1089     def __ne__(self, other):
1090         return not self == other
1091
1092     def __hash__(self):
1093         return hash((self.recipients, self.header))
1094
1095     def __len__(self):
1096         return len(self.recipients)
1097
1098     def __iter__(self):
1099         return iter(self.recipients)
1100
1101     def __contains__(self, item):
1102         return item in self.recipients
1103
1104     def __getitem__(self, index):
1105         return self.recipients[index]
1106
1107     def __setitem__(self, index, value):
1108         self.recipients[index] = value
1109
1110     def __delitem__(self, index):
1111         del self.recipients[index]
1112
1113     def __add__(self, other):
1114         if not isinstance(other, JweEnvelope):
1115             raise TypeError("Can only add another JweEnvelope object")
1116
1117         new_recipients = self.recipients + other.recipients
1118
1119         new_header = self.header
1120         if other.header:
1121             new_header = {**self.header, **other.header}
1122
1123         return JweEnvelope(new_recipients, new_header)
1124
1125     def __iadd__(self, other):
1126         if not isinstance(other, JweEnvelope):
1127             raise TypeError("Can only add another JweEnvelope object")
1128
1129         self.recipients += other.recipients
1130
1131         if other.header:
1132             self.header = {**other.header, **self.header}
1133
1134         return self
1135
1136     def __radd__(self, other):
1137         if not isinstance(other, JweEnvelope):
1138             raise TypeError("Can only add another JweEnvelope object")
1139
1140         new_recipients = other.recipients + self.recipients
1141
1142         new_header = other.header
1143         if self.header:
1144             new_header = {**other.header, **self.header}
1145
1146         return JweEnvelope(new_recipients, new_header)
1147
1148     def __rradd__(self, other):
1149         if not isinstance(other, JweEnvelope):
1150             raise TypeError("Can only add another JweEnvelope object")
1151
1152         self.recipients += other.recipients
1153
1154         if other.header:
1155             self.header = {**other.header, **self.header}
1156
1157         return self
1158
1159     def __sub__(self, other):
1160         if not isinstance(other, JweEnvelope):
1161             raise TypeError("Can only subtract another JweEnvelope object")
1162
1163         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1164
1165         new_header = self.header
1166         if other.header:
1167             new_header = {**self.header, **other.header}
1168
1169         return JweEnvelope(new_recipients, new_header)
1170
1171     def __isub__(self, other):
1172         if not isinstance(other, JweEnvelope):
1173             raise TypeError("Can only subtract another JweEnvelope object")
1174
1175         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1176
1177         if other.header:
1178             self.header = {**other.header, **self.header}
1179
1180         return self
1181
1182     def __eq__(self, other):
1183         if not isinstance(other, JweEnvelope):
1184             return False
1185
1186         if self.recipients != other.recipients:
1187             return False
1188
1189         if self.header != other.header:
1190             return False
1191
1192         return True
1193
1194     def __ne__(self, other):
1195         return not self == other
1196
1197     def __hash__(self):
1198         return hash((self.recipients, self.header))
1199
1200     def __len__(self):
1201         return len(self.recipients)
1202
1203     def __iter__(self):
1204         return iter(self.recipients)
1205
1206     def __contains__(self, item):
1207         return item in self.recipients
1208
1209     def __getitem__(self, index):
1210         return self.recipients[index]
1211
1212     def __setitem__(self, index, value):
1213         self.recipients[index] = value
1214
1215     def __delitem__(self, index):
1216         del self.recipients[index]
1217
1218     def __add__(self, other):
1219         if not isinstance(other, JweEnvelope):
1220             raise TypeError("Can only add another JweEnvelope object")
1221
1222         new_recipients = self.recipients + other.recipients
1223
1224         new_header = self.header
1225         if other.header:
1226             new_header = {**self.header, **other.header}
1227
1228         return JweEnvelope(new_recipients, new_header)
1229
1230     def __iadd__(self, other):
1231         if not isinstance(other, JweEnvelope):
1232             raise TypeError("Can only add another JweEnvelope object")
1233
1234         self.recipients += other.recipients
1235
1236         if other.header:
1237             self.header = {**other.header, **self.header}
1238
1239         return self
1240
1241     def __radd__(self, other):
1242         if not isinstance(other, JweEnvelope):
1243             raise TypeError("Can only add another JweEnvelope object")
1244
1245         new_recipients = other.recipients + self.recipients
1246
1247         new_header = other.header
1248         if self.header:
1249             new_header = {**other.header, **self.header}
1250
1251         return JweEnvelope(new_recipients, new_header)
1252
1253     def __rradd__(self, other):
1254         if not isinstance(other, JweEnvelope):
1255             raise TypeError("Can only add another JweEnvelope object")
1256
1257         self.recipients += other.recipients
1258
1259         if other.header:
1260             self.header = {**other.header, **self.header}
1261
1262         return self
1263
1264     def __sub__(self, other):
1265         if not isinstance(other, JweEnvelope):
1266             raise TypeError("Can only subtract another JweEnvelope object")
1267
1268         new_recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1269
1270         new_header = self.header
1271         if other.header:
1272             new_header = {**self.header, **other.header}
1273
1274         return JweEnvelope(new_recipients, new_header)
1275
1276     def __isub__(self, other):
1277         if not isinstance(other, JweEnvelope):
1278             raise TypeError("Can only subtract another JweEnvelope object")
1279
1280         self.recipients = [recipient for recipient in self.recipients if recipient not in other.recipients]
1281
1282         if other.header:
1283             self.header = {**other.header, **self.header}
1284
1285         return self
1286
1287     def __
```

```

217         )
218     )
219
220     # Use shared_secret as CEK
221     cek_bytes = shared_secret[:32]
222     cek = Key.from_secret_bytes(KeyAlg.C20P, cek_bytes) # Recreate CEK
223     from shared_secret!
224
225 else:
226     # Classical key (ED25519) - decode from base58 and convert to X25519
227     LOGGER.debug(f"Decoding classical ED25519 recipient key: {target_vk}
228     [:20]}...")
229
230     # Decode ED25519 public key from base58
231     ed_public_bytes = b58_to_bytes(target_vk)
232
233     # Convert ED25519 --> X25519 for ECDH
234     from aries_askar import crypto_box
235     target_xk = Key.from_public_bytes(KeyAlg.ED25519, ed_public_bytes).
236         convert_key(KeyAlg.X25519)
237
238 if sender_vk:
239     # Classical authcrypt
240     if isinstance(from_key, Key):
241         sender_xk = from_key.convert_key(KeyAlg.X25519)
242         enc_sender = crypto_box.crypto_box_seal(target_xk, sender_vk)
243         nonce = crypto_box.random_nonce()
244         enc_cek = crypto_box.crypto_box(target_xk, sender_xk, cek_bytes,
245             nonce)
246
247         wrapper.add_recipient(
248             JweRecipient(
249                 encrypted_key=enc_cek,
250                 header=OrderedDict([
251                     ("kid", target_vk),
252                     ("sender", b64url(enc_sender)),
253                     ("iv", b64url(nonce)),
254                 ]))
255             )
256         )
257     else:
258         raise ValueError("Classical authcrypt requires classical sender
259         key")
260
261 else:
262     # Classical anoncrypt
263     enc_cek = crypto_box.crypto_box_seal(target_xk, cek_bytes)
264     wrapper.add_recipient(
265         JweRecipient(
266             encrypted_key=enc_cek,
267             header={"kid": target_vk}
268         )
269     )
270
271
272 # 5. Set protected header
273 alg = "PQC-Authcrypt" if from_key else "PQC-Anoncrypt"
274 wrapper.set_protected(
275     OrderedDict([
276

```

```

269         ("enc", "xchacha20poly1305_ietf"),
270         ("typ", "JWM/1.0"),
271         ("alg", alg),
272     ])
273 )
274
275 # 6. Encrypt message with CEK
276 enc = cek.aead_encrypt(message_bytes, aad=wrapper.protected_bytes)
277 ciphertext, tag, nonce = enc.parts
278 wrapper.set_payload(ciphertext, nonce, tag)
279
280 # 7. Serialize to JSON
281 return wrapper.to_json().encode("utf-8")
282
283
284 async def unpack_message_pqc(
285     session: Session,
286     enc_message: bytes
287 ) -> Tuple[str, str, str]:
288     """Unpack a message with PQC support.
289
290     This function automatically detects whether the message was packed with
291     PQC or classical crypto based on the "alg" field in the JWE protected header.
292
293     Args:
294         session: Askar session for key lookups
295         enc_message: JWE-encoded message
296
297     Returns:
298         Tuple[str, str, str]: (message, sender_verkey, recipient_verkey)
299     """
300     from acapy_agent.utils.jwe import JweEnvelope
301     from acapy_agent.wallet.base import WalletError
302     from marshmallow import ValidationError
303
304     try:
305         wrapper = JweEnvelope.from_json(enc_message)
306     except ValidationError:
307         raise WalletError("Invalid packed message")
308
309     alg = wrapper.protected.get("alg")
310
311     # Detect PQC mode
312     if alg in ("PQC-Authcrypt", "PQC-Anoncrypt"):
313         LOGGER.info(f"Using PQC unpack mode: alg={alg}")
314         return await _unpack_pqc(session, wrapper)
315     elif alg in ("Authcrypt", "Anoncrypt"):
316         # Delegate to classical implementation
317         LOGGER.debug(f"Using classical unpack mode: alg={alg}")
318         from acapy_agent.askar.didcomm.v1 import unpack_message as unpack_classical
319         return await unpack_classical(session, enc_message)
320     else:
321         raise WalletError(f"Unsupported pack algorithm: {alg}")
322
323
324 async def _unpack_pqc(
325     session: Session,

```

```

326     wrapper
327 ) -> Tuple[str, str, str]:
328     """Unpack a PQC-encrypted message.
329
330     Algorithm:
331         1. Extract recipients from JWE
332         2. Try each recipient until we find one we can decrypt:
333             - Fetch our ML-KEM-768 secret key
334             - Decapsulate CEK using ML-KEM-768
335             - If authcrypt, verify signature with sender's ML-DSA-65
336         3. Decrypt message with CEK using ChaCha20-Poly1305
337         4. Return (message, sender_vk, recipient_vk)
338
339     Args:
340         session: Askar session for key lookups
341         wrapper: JweEnvelope object
342
343     Returns:
344         Tuple[str, str, str]: (message, sender_verkey, recipient_verkey)
345     """
346
347     from acapy_agent.wallet.base import WalletError
348     from acapy_agent.utils.jwe import b64url
349
350     alg = wrapper.protected.get("alg")
351     is_authcrypt = alg == "PQC-Authcrypt"
352
353     # Get liboqs for KEM operations
354     liboqs = get_liboqs()
355
356     # Parse recipients manually (don't use extract_pack_recipients - it's classical-
357     # only)
358     # PQC format: {"kid": "...", "sender": "...", "sig": "..."}
359     # Classical format: {"kid": "...", "sender": "...", "iv": "..."}
360     # NOTE: wrapper.recipients returns JweRecipient objects, not dicts
361     recips = {}
362     for recipient in wrapper.recipients:
363         kid = recipient.header.get("kid")
364         if kid:
365             recips[kid] = {
366                 "key": recipient.encrypted_key,
367                 "sender": recipient.header.get("sender"),
368                 "sig": recipient.header.get("sig"), # PQC: ML-DSA-65 signature
369                 "iv": recipient.header.get("iv"), # Classical: nonce
370                 "nonce": recipient.header.get("iv"), # Alias for compatibility
371             }
372
373     # Try to decrypt for each recipient we have a key for
374     cek_bytes, sender_vk, recip_vk = None, None, None
375
376     LOGGER.error("=" * 80)
377     LOGGER.error("[PQC UNPACK DEBUG] Starting recipient decryption loop")
378     LOGGER.error(f"[PQC UNPACK DEBUG] Found {len(recips)} recipients in JWE")
379     LOGGER.error(f"[PQC UNPACK DEBUG] Recipient KIDs: {list(recips.keys())}")
380
381     for recip_verkey, recip_data in recips.items():
382         LOGGER.error(f"[PQC UNPACK DEBUG] Trying recipient: {recip_verkey[:60] if len(
383             recip_verkey) > 60 else recip_verkey}...")

```

```

381
382     # Try to fetch our secret key
383     LOGGER.error(f"[PQC UNPACK DEBUG] Calling session.fetch_key({recip_verkey
384         [:60]}...)")
385     recip_key_entry = await session.fetch_key(recip_verkey)
386     LOGGER.error(f"[PQC UNPACK DEBUG] fetch_key result: {recip_key_entry}")
387
388     if recip_key_entry:
389         # Handle both KeyEntry wrapper (classical) and direct PQCKey
390         if isinstance(recip_key_entry, PQCKey):
391             recip_key = recip_key_entry # Already a PQCKey
392         else:
393             recip_key = recip_key_entry.key # KeyEntry wrapper
394
395         if isinstance(recip_key, PQCKey) and recip_key.algorithm == "ml-kem-768":
396             # PQC path: ML-KEM-768 decapsulation
397             secret_key = recip_key.get_secret_bytes()
398             ciphertext = recip_data["key"] # encrypted_key from JWE recipient
399
400         try:
401             # Decapsulate to get shared secret (used as CEK)
402             shared_secret = liboqs.kem_decapsulate(secret_key, ciphertext)
403             cek_bytes = shared_secret[:32] # ChaCha20 needs 256-bit key
404
405             # Check for authcrypt signature
406             if is_authcrypt:
407                 sig = recip_data.get("sig")
408                 sender_vk_b64 = recip_data.get("sender")
409
410                 if not sig or not sender_vk_b64:
411                     raise WalletError("Authcrypt message missing signature or
412                         sender")
413
414                 # Decode sender verkey and signature
415                 import base64
416                 from acapy_agent.wallet.util import b58_to_bytes
417
418                 sender_vk = base64.urlsafe_b64decode(sender_vk_b64 + "==").
419                     decode("utf-8")
420                 signature = base64.urlsafe_b64decode(sig + "==")
421
422                 # Extract sender's ML-DSA-65 public key directly from message
423                 # (Like classical ED25519: don't fetch from wallet, use from
424                 # message!)
425                 sender_public = b58_to_bytes(sender_vk)
426
427                 # Verify signature
428                 LOGGER.error(f"[PQC UNPACK DEBUG] Verifying ML-DSA-65
signature...")
429                 LOGGER.error(f"[PQC UNPACK DEBUG]    sender_vk (base58): {
430                     sender_vk[:60]}...")
431                 LOGGER.error(f"[PQC UNPACK DEBUG]    sender_public (bytes): {
432                     len(sender_public)} bytes")
433                 if not liboqs.ml_dsa_verify(sender_public, ciphertext,
434                     signature):
435                     raise WalletError("Invalid ML-DSA-65 signature in
authcrypt")

```

```

429         LOGGER.error(f"[PQC UNPACK DEBUG] Signature verified!")
430
431     recip_vk = recip_verkey
432     LOGGER.error(f"[PQC UNPACK DEBUG] SUCCESS! recip_vk set to: {recip_vk[:60] if len(recip_vk) > 60 else recip_vk}")
433     break # Success!
434
435 except Exception as e:
436     LOGGER.error(f"[PQC UNPACK DEBUG] Decryption failed: {e}")
437     LOGGER.debug(f"Failed to decrypt for recipient {recip_verkey[:20]}...: {e}")
438     continue
439
440 else:
441     # Classical key - should not happen in pure PQC mode
442     LOGGER.warning(f"Recipient {recip_verkey[:20]}... is classical, " f"trying classical crypto_box")
443
444     # Fallback to classical decryption
445     from acapy_agent.askar.didcomm.v1 import _extract_payload_key
446     try:
447         cek_bytes, sender_vk = _extract_payload_key(recip_data, recip_key)
448         recip_vk = recip_verkey
449         break
450     except Exception as e:
451         LOGGER.error(f"[PQC UNPACK DEBUG] Classical decryption failed: {e}")
452         LOGGER.debug(f"Classical decrypt failed for {recip_verkey[:20]}...: {e}")
453         continue
454     else:
455         LOGGER.error(f"[PQC UNPACK DEBUG] No key found for recipient {recip_verkey[:60] if len(recip_verkey) > 60 else recip_verkey}")
456
457     LOGGER.error(f"[PQC UNPACK DEBUG] Loop finished. recip_vk = {recip_vk}")
458     LOGGER.error("=" * 80)
459
460 if not cek_bytes:
461     raise WalletError(f"No corresponding recipient key found in {tuple(recips.keys())}")
462
463 if not sender_vk and is_authcrypt:
464     raise WalletError("Sender public key not provided for Authcrypt message")
465
466     # Decrypt message with CEK
467     cek = Key.from_secret_bytes(KeyAlg.C20P, cek_bytes)
468     message = cek.aead_decrypt(
469         wrapper.ciphertext,
470         nonce=wrapper.iv,
471         tag=wrapper.tag,
472         aad=wrapper.protected_bytes,
473     )
474
475
476     # Return in ACA-Py format: (message, from_verkey, to_verkey)
477     # from_verkey = sender, to_verkey = recipient
478     return message.decode("utf-8"), sender_vk, recip_vk

```

Anhang 4.1.7: monkey_patches.py

Listing A-19

pqc_didpeer4_fm - monkey_patches.py


```

104         await attach.data.sign(my_info.verkey, wallet)
105
106     LOGGER.info("Attachment re-signed with did:peer:4 authentication key")
107 else:
108     LOGGER.debug("  Keys match, no re-signing needed")
109
110 return did, attach
111
112
113 def apply_all_patches():
114     """Apply all monkey patches for transparent PQC integration.
115
116     This function:
117     1. Patches BaseConnectionManager.create_did_peer_4 with PQC version
118     2. Patches BaseConnectionManager._extract_key_material_in_base58_format for PQC
119     3. Patches BaseConnectionManager.long_did_peer_4_to_short to preserve key_type
120     4. Patches BaseConnectionManager.long_did_peer_to_short to handle short-form DIDs
121     5. Patches DIDXManager._qualified_did_with_fallback for correct PQC signatures
122     6. Patches BaseConnectionManager.record_keys_for_resolvable_did for PQC key
123         storage
124     7. Patches BaseConnectionManager.resolve_inbound_connection with DEBUG logging
125     8. Patches BaseConnectionManager.find_connection to handle long/short form my_did
126     9. Extends PEER4 DID method to support ML-DSA-65 and ML-KEM-768 key types
127
128     Called during plugin setup.
129     """
130
131     # 1. Patch BaseConnectionManager.create_did_peer_4 with CRYPTO-AGILE wrapper
132     # KRYPTO-AGILITÄT: Conditional wrapper that checks metadata["key_type"]
133     # - If "ed25519" --> Delegates to _original_create_did_peer_4 (pre-plugin behavior
134     # )
135     # - Otherwise --> Uses ML-DSA-65 + ML-KEM-768 (PQC)
136     # This enables mixed environments where ED25519 and PQC can coexist!
137     BaseConnectionManager.create_did_peer_4 = create_did_peer_4_conditional_pqc
138
139     # 2. Patch BaseConnectionManager._extract_key_material_in_base58_format
140     # CRITICAL: This allows out-of-band invitation creation to work with PQC keys
141     # by accepting ml-dsa-65-pub and ml-kem-768-pub multicodecs
142     BaseConnectionManager._extract_key_material_in_base58_format = staticmethod(
143         _extract_key_material_in_base58_format_pqc
144     )
145
146     # 3. Patch BaseConnectionManager.long_did_peer_4_to_short
147     # CRITICAL: This preserves PQC key_type when converting long-form to short-form,
148     # eliminating ED25519 "ghost DIDs" in connections!
149     BaseConnectionManager.long_did_peer_4_to_short = long_did_peer_4_to_short_pqc
150
151     # 4. Patch BaseConnectionManager.long_did_peer_to_short
152     # CRITICAL FIX: This prevents double-conversion of short-form DIDs during
153     # connection lookup, which was causing "No connection found" errors during
154     # credential offer handling!
155     BaseConnectionManager.long_did_peer_to_short = long_did_peer_to_short_pqc
156
157     # 5. Patch DIDXManager._qualified_did_with_fallback for correct PQC signatures
158     # CRITICAL: This ensures did:peer:4 attachments are signed with the DID's
159     # authentication key, not the invitation_key!
160     DIDXManager._qualified_did_with_fallback = _qualified_did_with_fallback_pqc

```

```

159
160     # 6. Patch BaseConnectionManager.record_keys_for_resolvable_did
161     # CRITICAL FIX: Stores BOTH keys (ML-DSA-65 auth + ML-KEM-768 keyAgr) in DID_KEY
162     # records
163     # This enables find_did_for_key(sender_verkey) to work during message handling!
164     # Original only stored recipientKeys (ML-KEM-768), causing connection lookup to
165     # fail
166     # when searching for sender's authentication key (ML-DSA-65)
167     BaseConnectionManager.record_keys_for_resolvable_did =
168         record_keys_for_resolvable_did_pqc
169
170
171     # 7. Patch BaseConnectionManager.resolve_inbound_connection
172     # DEBUG PATCH: Adds extensive logging to diagnose why recipient_did is None
173     BaseConnectionManager.resolve_inbound_connection = resolve_inbound_connection_pqc
174
175     # 8. Patch BaseConnectionManager.find_connection
176     # CRITICAL FIX for credential offer handling: Handles both long and short forms of
177     # my_did
178     # This fixes "No connection found" errors during credential issuance when:
179     # - Connection stored with my_did = LONG-FORM (during DID Exchange)
180     # - Credential offer uses my_did = SHORT-FORM (from wallet.
181         get_local_did_for_verkey)
182     BaseConnectionManager.find_connection = find_connection_pqc
183
184     # 9. Extend PEER4 key types for validation (optional but recommended)
185     if ML_DSA_65 not in PEER4._supported_key_types:
186         PEER4._supported_key_types.extend([ML_DSA_65, ML_KEM_768])

```

Anhang 4.1.8: base_manager_patch.py

Listing A-20

```

pqc_didpeer4_fm - base_manager_patch.py
1 """Patch for acapy_agent/connections/base_manager.py to use PQC keys.
2
3 This patch completely replaces BaseConnectionManager.create_did_peer_4 to use
4 ML-DSA-65 + ML-KEM-768 instead of ED25519, eliminating the creation of ED25519
5 "ghost DIDs" in the wallet.
6
7 NO modifications to acapy_agent source code needed - all changes are in the plugin!
8 """
9
10 import logging
11 from typing import Dict, List, Optional, Sequence
12
13 from did_peer_4 import encode
14 from did_peer_4.input_doc import input_doc_from_keys_and_services, KeySpec as
15    KeySpec_DP4
16
17 from acapy_agent.wallet.base import BaseWallet
18 from acapy_agent.wallet.did_info import DIDInfo
19 from acapy_agent.wallet.did_method import PEER4
20 from acapy_agent.protocols.coordinate_mediation.v1_0.models.mediation_record import
21     MediationRecord

```

```

20| from acapy_agent.did.did_key import DIDKey
21| from acapy_agent.wallet.util import bytes_to_b58, b64_to_bytes
22| from acapy_agent.connections.base_manager import BaseConnectionManagerError
23| from acapy_agent.utils.multiformats import multibase, multicodec
24|
25| from .key_types import ML_DSA_65, ML_KEM_768
26| from .pqc_multikey import key_info_to_multikey
27|
28| LOGGER = logging.getLogger(__name__)
29|
30|
31| async def create_did_peer_4_pqc_complete(
32|     self,
33|     svc_endpoints: Optional[Sequence[str]] = None,
34|     mediation_records: Optional[List[MediationRecord]] = None,
35|     metadata: Optional[Dict] = None,
36| ) -> DIDInfo:
37|     """Create a did:peer:4 DID for a connection using PQC keys.
38|
39|     This is a COMPLETE replacement for BaseConnectionManager.create_did_peer_4
40|     that uses ML-DSA-65 + ML-KEM-768 instead of ED25519.
41|
42|     CRITICAL DIFFERENCE from original:
43|     - Line 165 in original: `key = await wallet.create_key(ED25519)`
44|     - This version: Creates ML-DSA-65 (authentication) + ML-KEM-768 (key agreement)
45|
46|     This eliminates the ED25519 "ghost DIDs" that were previously created!
47|
48| Args:
49|     self: BaseConnectionManager instance
50|     svc_endpoints: Custom endpoints for the DID Document
51|     mediation_records: Records for mediation that contain routing keys and
52|         endpoint
53|     metadata: Additional metadata for the DID
54|
55| Returns:
56|     DIDInfo: The new PQC did:peer:4 DID
57| """
58|
59| # Extract routing keys from mediation (same as original)
60| routing_keys: List[str] = []
61| if mediation_records:
62|     for mediation_record in mediation_records:
63|         (
64|             mediator_routing_keys,
65|             endpoint,
66|         ) = await self._route_manager.routing_info(
67|             self._profile, mediation_record
68|         )
69|         routing_keys = [*routing_keys, *(mediator_routing_keys or [])]
70|         if endpoint:
71|             svc_endpoints = [endpoint]
72|
73| # Build DIDComm v1 services (same as original)
74| services = []
75| for index, endpoint in enumerate(svc_endpoints or []):
    services.append(
        {

```

```

76         "id": f"#didcomm-{index}",
77         "type": "did-communication",
78         "recipientKeys": ["#key-1"], # ML-KEM-768 key for encryption (
79             key_specs[1] --> #key-1)
80         "routingKeys": routing_keys,
81         "serviceEndpoint": endpoint,
82         "priority": index,
83     }
84 )
85
86 async with self._profile.session() as session:
87     wallet = session.inject(BaseWallet)
88
89 # =====
90 # PQC MODIFICATION: Create TWO keys instead of one ED25519 key
91 # =====
92
93 # 1. Create ML-DSA-65 key for authentication/assertion (signatures)
94 sig_key = await wallet.create_key(ML_DSA_65)
95 sig_multikey = key_info_to_multikey(sig_key)
96
97 # 2. Create ML-KEM-768 key for key agreement (encryption)
98 kem_key = await wallet.create_key(ML_KEM_768)
99 kem_multikey = key_info_to_multikey(kem_key)
100
101 LOGGER.info(f"Created PQC keys for did:peer:4:")
102 LOGGER.info(f" - ML-DSA-65 (auth): {sig_multikey[:40]}...")
103 LOGGER.info(f" - ML-KEM-768 (keyAgr): {kem_multikey[:40]}...")
104
105 # 3. Create KeySpec objects for did:peer:4 (same as pqc_peer4_creator.py)
106 # NOTE: Order matters! did-peer-4 numbers from 0: key_specs[0] --> #key-0,
107     key_specs[1] --> #key-1
108 key_specs = [
109    KeySpec_DP4(
110         multikey=sig_multikey,
111         relationships=["authentication", "assertionMethod"], # --> #key-0
112     ),
113    KeySpec_DP4(
114         multikey=kem_multikey,
115         relationships=[ "keyAgreement"], # --> #key-1 (used in recipientKeys!)
116     ),
117 ]
118
119 # 4. Generate did:peer:4 long form (same structure as original)
120 input_doc = input_doc_from_keys_and_services(
121     keys=key_specs,
122     services=services
123 )
124 did = encode(input_doc)
125
126 LOGGER.info(f"Generated PQC did:peer:4: {did[:80]}...")
127
128 # 5. Create metadata with PQC markers
129 did_metadata = metadata if metadata else {}
130 did_metadata.update({
131     "pqc_enabled": True,
132     "signature_algorithm": "ml-dsa-65",
133 })

```

```

131         "key_agreement_algorithm": "ml-kem-768",
132         "kem_key_kid": f"{did}#key-1", # KEM key is key_specs[1] --> #key-1
133         "kem_verkey": kem_key.verkey, # CRITICAL: Store KEM verkey for connection
134             lookup!
135         "plugin": "pqc_didpeer4_fm",
136         "version": "0.1.0",
137     })
138
139     LOGGER.error(f"STORING PQC DID WITH KEM_VERKEY IN METADATA:")
140     LOGGER.error(f"    DID: {did[:60]}...")
141     LOGGER.error(f"    Primary verkey (ML-DSA-65): {sig_key.verkey[:30]}...")
142     LOGGER.error(f"    KEM verkey (ML-KEM-768): {kem_key.verkey[:30]}...")
143     LOGGER.error(f"    Metadata keys: {list(did_metadata.keys())}")
144
145     # 6. Create DIDInfo with ML-DSA-65 as primary key type
146     did_info = DIDInfo(
147         did=did,
148         method=PEER4,
149         verkey=sig_key.verkey, # ML-DSA-65 verkey
150         metadata=did_metadata,
151         key_type=ML_DSA_65, # NOT ED25519!
152     )
153
154     # 7. Store DID in wallet
155     await wallet.store_did(did_info)
156
157     # 8. Assign Key IDs - did-peer-4 numbers from 0!
158     await wallet.assign_kid_to_key(sig_key.verkey, f"{did}#key-0") # key_specs[0]
159         --> #key-0
160     await wallet.assign_kid_to_key(kem_key.verkey, f"{did}#key-1") # key_specs[1]
161         --> #key-1
162
163     LOGGER.info(f"Stored PQC did:peer:4 in wallet with key IDs assigned")
164
165     return did_info
166
167
168 async def create_did_peer_4_conditional_pqc(
169     self,
170     svc_endpoints: Optional[Sequence[str]] = None,
171     mediation_records: Optional[List[MediationRecord]] = None,
172     metadata: Optional[Dict] = None,
173 ) -> DIDInfo:
174     """Conditional wrapper: Use PQC or original ED25519 based on metadata.
175     """
176     from acapy_agent.connections.base_manager import BaseConnectionManager
177
178     # Check: Wurde ED25519 explizit gewünscht?
179     if metadata and metadata.get("key_type") == "ed25519":
180         LOGGER.info("Kryptoagilität: ED25519 gewünscht --> Nutze Original-ACA-Py (KEIN
181             Plugin-Patch)")
182         # Delegation an ursprüngliche ACA-Py-Implementierung ohne Plugin-Eingriff
183         # Die Original-Funktion wurde gesichert bevor das Plugin sie überschrieben hat
184         from . import monkey_patches
185         if hasattr(monkey_patches, '_original_create_did_peer_4'):
186             return await monkey_patches._original_create_did_peer_4(
187                 self, svc_endpoints, mediation_records, metadata
188             )

```

```

184     else:
185         # Fallback: Rufe die aktuelle Implementierung auf (sollte nicht passieren)
186         LOGGER.warning("Original create_did_peer_4 nicht gefunden, nutze aktuelle
187             Implementierung")
188         return await BaseConnectionManager.create_did_peer_4(
189             self, svc_endpoints, mediation_records, metadata
190         )
191
192     # Default: Nutze PQC
193     LOGGER.info("Kryptoagilität: PQC-Algorithmen (ML-DSA-65 + ML-KEM-768)")
194     return await create_did_peer_4_pqc_complete(
195         self, svc_endpoints, mediation_records, metadata
196     )
197
198     def _extract_key_material_in_base58_format_pqc(method) -> str:
199         """Patched version of BaseConnectionManager._extract_key_material_in_base58_format
200             .
201
202             This patch extends the original method to accept PQC multicodecs (ML-DSA-65 and ML
203                 -KEM-768)
204             in addition to the classical ED25519 multicodec.
205
206             CRITICAL: This method is called during out-of-band invitation creation when
207                 resolving
208             recipient keys from did:peer:4 DID Documents. The original only accepts ed25519-
209                 pub,
210             causing BaseConnectionManagerError when encountering PQC keys.
211
212             This patched version:
213             1. Maintains full backward compatibility with all existing key types
214             2. Adds support for ml-dsa-65-pub and ml-kem-768-pub multicodecs
215             3. Returns base58-encoded key material for all supported key types
216
217             Args:
218                 method: VerificationMethod from DID Document
219
220             Returns:
221                 str: Base58-encoded key material
222
223             Raises:
224                 BaseConnectionManagerError: If key type or multicodec is not supported
225             """
226
227             from pydid.verification_method import (
228                 Ed25519VerificationKey2018,
229                 Ed25519VerificationKey2020,
230                 JsonWebKey2020,
231                 Multikey,
232             )
233
234             if isinstance(method, Ed25519VerificationKey2018):
235                 return method.material
236             elif isinstance(method, Ed25519VerificationKey2020):
237                 raw_data = multibase.decode(method.material)
238                 if len(raw_data) == 32: # No multicodec prefix
239                     return bytes_to_b58(raw_data)
240                 else:
241                     codec, key = multicodec.unwrap(raw_data)

```

```

236         if codec == multicodec.multicodec("ed25519-pub"):
237             return bytes_to_b58(key)
238         else:
239             raise BaseConnectionManagerError(
240                 f"Key type {type(method).__name__} "
241                 f"with multicodec value {codec} is not supported"
242             )
243
244     elif isinstance(method, JsonWebKey2020):
245         if method.public_key_jwk.get("kty") == "OKP":
246             return bytes_to_b58(b64_to_bytes(method.public_key_jwk.get("x"), True))
247         else:
248             raise BaseConnectionManagerError(
249                 f"Key type {type(method).__name__} "
250                 f"with kty {method.public_key_jwk.get('kty')} is not supported"
251             )
252     elif isinstance(method, Multikey):
253         codec, key = multicodec.unwrap(multibase.decode(method.material))
254
255     # PQC EXTENSION: Accept ML-DSA-65 and ML-KEM-768 in addition to ED25519
256     accepted_codecs = [
257         multicodec.multicodec("ed25519-pub"),           # Classical
258         multicodec.multicodec("ml-dsa-65-pub"),        # PQC signature
259         multicodec.multicodec("ml-kem-768-pub"),        # PQC key agreement
260     ]
261
262     if codec not in accepted_codecs:
263         raise BaseConnectionManagerError(
264             f"Expected ed25519-pub, ml-dsa-65-pub, or ml-kem-768-pub multicodec,
265             got: {codec}"
266         )
267
268     LOGGER.debug(f"Extracting key material from Multikey with codec: {codec.name}")
269     return bytes_to_b58(key)
270 else:
271     raise BaseConnectionManagerError(
272         f"Key type {type(method).__name__} is not supported"
273     )
274
275 async def long_did_peer_4_to_short_pqc(self, long_dp4: str) -> str:
276     """Convert did:peer:4 long format to short format and store in wallet (PQC version
277     )."""
278
279     This is a patched version of BaseConnectionManager.long_did_peer_4_to_short that:
280     1. Preserves the key_type from the original long-form DID (ML-DSA-65 instead of
281         hardcoded ED25519)
282     2. Preserves metadata from the original DID
283     3. Ensures quantum-safe connections
284
285     CRITICAL DIFFERENCE from original (base_manager.py:112):
286     - Original: 'key_type=ED25519' (hardcoded)
287     - This version: 'key_type=long_dp4_info.key_type' (preserved from long-form)
288
289     This eliminates the ED25519 "ghost DIDs" in connections!
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
889

```

```

289     Args:
290         self: BaseConnectionManager instance
291         long_dp4: Long-form did:peer:4 DID
292
293     Returns:
294         str: Short-form did:peer:4 DID
295     """
296     from did_peer_4 import long_to_short
297
298     async with self._profile.session() as session:
299         wallet = session.inject(BaseWallet)
300         long_dp4_info = await wallet.get_local_did(long_dp4)
301
302         short_did_peer_4 = long_to_short(long_dp4)
303
304         # CRITICAL: Use key_type from original DID, NOT hardcoded ED25519!
305         did_info = DIDInfo(
306             did=short_did_peer_4,
307             method=PEER4,
308             verkey=long_dp4_info.verkey,
309             metadata=long_dp4_info.metadata or {}, # Preserve metadata
310             key_type=long_dp4_info.key_type, # Preserve key_type (ML-DSA-65)
311         )
312
313         async with self._profile.session() as session:
314             wallet = session.inject(BaseWallet)
315             await wallet.store_did(did_info)
316
317             LOGGER.info(f"Converted Long-Form to Short-Form (key_type preserved: {long_dp4_info.key_type})")
318             LOGGER.debug(f"    Long: {long_dp4[:80]}...")
319             LOGGER.debug(f"    Short: {short_did_peer_4}")
320
321         return did_info.did
322
323
324     async def record_keys_for_resolvable_did_pqc(self, did: str):
325         """Store verkeys for lookup against a DID (PQC version that stores BOTH keys).
326
327         This is an EXTENDED version of BaseConnectionManager.
328         record_keys_for_resolvable_did
329         that stores BOTH recipient keys (ML-KEM-768) AND authentication keys (ML-DSA-65).
330
331         CRITICAL FIX for connection lookup during message handling:
332         - Original (base_manager.py:536-543): Only stores recipientKeys from DIDComm
333             service
334         - For classical did:peer:4: recipientKeys = [ED25519] --> Stored
335         - For PQC did:peer:4: recipientKeys = [ML-KEM-768 (#key-1)] --> Stored
336             BUT authentication key (ML-DSA-65 #key-0) is NOT stored!
337
338         Problem:
339         - DIDComm messages have sender_verkey = sender's authentication key (ML-DSA-65)
340         - 'find_did_for_key(sender_verkey)' looks up this key in DID_KEY records
341         - For PQC: ML-DSA-65 was never stored --> NOT FOUND! --> Connection lookup fails!
342
343         This patch:
344         1. Stores recipientKeys (original behavior - KEM key)

```

```

343     2. ALSO stores authentication keys (ML-DSA-65 for PQC)
344     3. Enables connection lookup for BOTH keys
345
346     Args:
347         self: BaseConnectionManager instance
348         did: The DID for which to record keys
349     """
350
351     from acapy_agent.resolver.did_resolver import DIDResolver
352
353     LOGGER.error(f"[PQC] ===== record_keys_for_resolvable_did_pqc CALLED =====")
354     LOGGER.error(f"[PQC]    DID: {did[:80]}...")
355
356     doc, didcomm_services = await self.resolve_didcomm_services(did)
357
358     # 1. Store recipient keys from DIDComm services (original behavior)
359     LOGGER.error(f"[PQC]    Found {len(didcomm_services)} DIDComm services")
360     for service in didcomm_services:
361         recips, _ = await self.verification_methods_for_service(doc, service)
362         LOGGER.error(f"[PQC]    Service has {len(recips)} recipient keys")
363         for recip in recips:
364             recip_key = self._extract_key_material_in_base58_format(recip)
365             LOGGER.error(f"[PQC]    Storing RECIPIENT key: {recip_key[:30]}... --> DID:
366                         {did[:60]}...")
367             await self.add_key_for_did(did, recip_key)
368
369     # 2. ALSO store authentication keys (NEW for PQC support!)
370     # This enables find_did_for_key(sender's ML-DSA-65) to work
371     if doc.authentication:
372         LOGGER.error(f"[PQC]    Found {len(doc.authentication)} authentication keys in
373                         DID Document")
374         resolver = self._profile.inject(DIDResolver)
375         for auth_ref in doc.authentication:
376             try:
377                 auth_method = await resolver.dereference_verification_method(
378                     self._profile, auth_ref, document=doc
379                 )
380                 auth_key = self._extract_key_material_in_base58_format(auth_method)
381                 LOGGER.error(f"[PQC] Storing AUTHENTICATION key: {auth_key[:30]}...
382                               --> DID: {did[:60]}...")
383                 await self.add_key_for_did(did, auth_key)
384             except Exception as e:
385                 LOGGER.error(f"[PQC] Could not store authentication key: {e}")
386                 # Don't fail if we can't store auth key (not critical)
387                 pass
388         else:
389             LOGGER.error(f"[PQC] No authentication keys found in DID Document!")
390
391     LOGGER.error(f"[PQC] ===== record_keys_for_resolvable_did_pqc DONE =====")
392
393     def long_did_peer_to_short_pqc(self, long_did: str) -> str:
394         """Convert did:peer:4 long format to short format and return (PQC-safe version).
395
396         This is a patched version of BaseConnectionManager.long_did_peer_to_short that
397         handles BOTH long-form and short-form DIDs correctly.
398
399         CRITICAL FIX for connection lookup during message handling:

```

```

397     - Original method (base_manager.py:93-97) blindly converts any did:peer:4 to short
398     - But after DID Exchange, DIDs are stored in SHORT-FORM in the wallet
399     - When 'find_connection' (line 877) looks up a connection, it assumes 'their_did'
400         is LONG-FORM and tries to convert it to short (line 900)
401     - If 'their_did' is ALREADY short-form, converting it again produces a WRONG hash
402     - Connection lookup fails --> "No connection found" error in credential offer
403         handler!
404
404 This patch:
405 1. Detects if the DID is already in short-form (no multibase-encoded DID Document
406     suffix)
407 2. Returns it unchanged if short-form
408 3. Converts if long-form (original behavior)
409
409 Short-form vs Long-form detection:
410 - Short-form: did:peer:4zQmZMkY... (just the hash, no suffix after 4th ":")
411 - Long-form: did:peer:4zQmZMkY:z25g... (hash + ":" + multibase-encoded doc)
412 - Count colons: SHORT has 2 colons ("did:peer:4..."), LONG has 3+ ("did:peer:4...:
413     z...")
414
414 Args:
415     self: BaseConnectionManager instance
416     long_did: did:peer:4 DID (can be long-form OR short-form)
417
418 Returns:
419     str: Short-form did:peer:4 DID
420 """
421 from did_peer_4 import long_to_short
422
423 # Check if already short-form by counting colons
424 # Short-form: "did:peer:4zQm..." --> 2 colons
425 # Long-form: "did:peer:4zQm...:z25g..." --> 3 colons
426 colon_count = long_did.count(":")
427 if colon_count == 2:
428     LOGGER.debug(f"DID is already short-form (2 colons), returning unchanged: {long_did[:60]}...")
429     return long_did
430
431 # Long-form --> convert to short
432 LOGGER.debug(f"Converting long-form to short-form ({colon_count} colons): {long_did[:60]}...")
433 short_did_peer = long_to_short(long_did)
434 LOGGER.debug(f" Result: {short_did_peer}")
435 return short_did_peer
436
437
438 async def resolve_inbound_connection_pqc(self, receipt):
439     """PQC-aware version of resolve_inbound_connection with sender-based lookup.
440
441     CRITICAL FIX for PQC multi-connection scenario:
442     - Problem: wallet.get_local_did_for_verkey(recipient_verkey) can return WRONG DID
443         when multiple connections exist (each with different ML-KEM-768 keys)
444     - The wallet patch searches ALL DIDs and returns the FIRST match, which might be
445         from a DIFFERENT connection (e.g., verifier connection instead of issuer
446         connection)
447
447 Solution: Prefer sender-based connection lookup over recipient-based:

```

```

448     1. Resolve sender_did from sender_verkey (their authentication key)
449     2. Find connection by their_did (sender_did) - this uniquely identifies the
450        connection
451     3. Only fall back to recipient_did lookup if sender-based lookup fails
452
453     This matches ED25519 workflow where connections are resolved from sender identity.
454     """
455
456     from acapy_agent.core.error import BaseError
457     from acapy_agent.storage.error import StorageNotFoundError
458     from acapy_agent.wallet.error import WalletNotFoundError
459
460     # InjectionError might be from config.injection_context
461     try:
462         from acapy_agent.config.injection_context import InjectionError
463     except ImportError:
464         # Fallback: use BaseError
465         InjectionError = BaseError
466
467     LOGGER.error("==" * 80)
468     LOGGER.error("[PQC FIX] resolve_inbound_connection CALLED")
469     LOGGER.error(f"[PQC FIX]   sender_verkey: {receipt.sender_verkey[:30]} if receipt."
470                  "sender_verkey else 'None'....")
471     LOGGER.error(f"[PQC FIX]   recipient_verkey: {receipt.recipient_verkey[:30]} if"
472                  "receipt.recipient_verkey else 'None'....")
473
474     # Step 1: Resolve sender_did from sender_verkey (CRITICAL for PQC!)
475     receipt.sender_did = None
476     if receipt.sender_verkey:
477         try:
478             LOGGER.error(f"[PQC FIX] Looking up sender_did from sender_verkey (their"
479                         "auth key)....")
480             receipt.sender_did = await self.find_did_for_key(receipt.sender_verkey)
481             LOGGER.error(f"[PQC FIX] Found sender_did (their_did): {receipt.sender_did"
482                         "[::60]}....")
483         except StorageNotFoundError as e:
484             LOGGER.error(f"[PQC FIX] sender_did NOT found: (e)")
485             pass
486
487     # Step 2: Try sender-based connection lookup FIRST (NEW for PQC!)
488     if receipt.sender_did:
489         LOGGER.error(f"[PQC FIX] TRYING SENDER-BASED LOOKUP (their_did={receipt."
490                     "sender_did[:60]}....)")
491         try:
492             connection = await self.find_connection(
493                 their_did=receipt.sender_did,
494                 my_did=None,  # Don't filter by my_did yet!
495                 parent_thread_id=receipt.parent_thread_id,
496                 auto_complete=True
497             )
498             if connection:
499                 LOGGER.error(f"[PQC FIX] Found connection via SENDER lookup!")
500                 LOGGER.error(f"[PQC FIX]   Connection: {connection.connection_id}")
501                 LOGGER.error(f"[PQC FIX]   their_did: {connection.their_did}")
502                 LOGGER.error(f"[PQC FIX]   my_did: {connection.my_did}")
503
504                 # Set recipient_did from the connection's my_did (correct one!)
505                 receipt.recipient_did = connection.my_did

```

```

499         LOGGER.error(f"[PQC FIX] Set recipient_did from connection: {receipt
500             .recipient_did[:60]}...")
501         LOGGER.error("=" * 80)
502         return connection
503     except Exception as e:
504         LOGGER.error(f"[PQC FIX] Sender-based lookup failed: {e}")
505
506     # Step 3: Fall back to recipient-based lookup (original behavior)
507     LOGGER.error(f"[PQC FIX] Falling back to RECIPIENT-BASED lookup...")
508     if receipt.recipient_verkey:
509         try:
510             LOGGER.error(f"[PQC FIX] Getting wallet and calling
511                 get_local_did_for_verkey...")
512             async with self._profile.session() as session:
513                 wallet = session.inject(BaseWallet)
514                 my_info = await wallet.get_local_did_for_verkey(
515                     receipt.recipient_verkey
516                 )
517                 LOGGER.error(f"[PQC FIX] Got my_info.did: {my_info.did[:60]}...")
518                 receipt.recipient_did = my_info.did
519                 if "posted" in my_info.metadata and my_info.metadata["posted"] is True:
520                     receipt.recipient_did_public = True
521                     LOGGER.error(f"[PQC FIX] Set recipient_did_public = True")
522                 except InjectionError as e:
523                     LOGGER.error(f"[PQC FIX] InjectionError: {e}")
524                     self._logger.warning(
525                         "Cannot resolve recipient verkey, no wallet defined by context: %s",
526                         receipt.recipient_verkey,
527                     )
528                 except WalletNotFoundError as e:
529                     LOGGER.error(f"[PQC FIX] WalletNotFoundError: {e}")
530                     self._logger.debug(
531                         "No corresponding DID found for recipient verkey: %s",
532                         receipt.recipient_verkey,
533                     )
534
535             LOGGER.error(f"[PQC FIX] Final sender_did: {receipt.sender_did[:60] if receipt.
536                 sender_did else 'None'}...")
537             LOGGER.error(f"[PQC FIX] Final recipient_did: {receipt.recipient_did[:60] if
538                 receipt.recipient_did else 'None'}...")
539             LOGGER.error("[PQC FIX] Calling find_connection with both DIDs...")
540
541             result = await self.find_connection(
542                 receipt.sender_did, receipt.recipient_did, receipt.parent_thread_id, True
543             )
544
545             return result
546
547
548     async def find_connection_pqc(
549         self,
550         their_did: Optional[str],
551         my_did: Optional[str] = None,

```

```

552     parent_thread_id: Optional[str] = None,
553     auto_complete=False,
554 ) :
555     """PQC-aware version of find_connection that handles both long and short forms of
556     my_did.
557
558     CRITICAL FIX for credential offer handling:
559     - Connection is stored with my_did = LONG-FORM during DID Exchange
560     - Credential offer processing uses my_did = SHORT-FORM (from wallet.
561         get_local_did_for_verkey)
562     - Original find_connection only queries with the exact my_did provided --> NOT
563         FOUND!
564
565     This patch:
566     1. Converts my_did to both long and short forms (if did:peer:4)
567     2. Queries database with: (their_did IN (long, short)) AND (my_did IN (long, short
568         ))
569     3. Returns the connection if found
570
571     This matches the existing logic for their_did in base_manager.py:897-907.
572
573     Args:
574         self: BaseConnectionManager instance
575         their_did: Their DID
576         my_did: My DID (can be long or short form for did:peer:4)
577         parent_thread_id: Parent thread ID
578         auto_complete: Should this connection automatically be promoted to active
579
580     Returns:
581         The located 'ConnRecord', if any
582     """
583
584     from acapy_agent.storage.error import StorageNotFoundError
585     from acapy_agent.connections.models.conn_record import ConnRecord
586     from acapy_agent.protocols.discovery.v2_0.manager import V20DiscoveryMgr
587     from did_peer_4 import long_to_short
588
589     LOGGER.error("=" * 80)
590     LOGGER.error("[PQC FIX] find_connection_pqc CALLED")
591     LOGGER.error(f"[PQC FIX]    their_did: {their_did[:60]} if their_did else 'None'...")
592     LOGGER.error(f"[PQC FIX]    my_did: {my_did[:60]} if my_did else 'None'...")
593     LOGGER.error(f"[PQC FIX]    parent_thread_id: {parent_thread_id}")
594
595     connection = None
596     if their_did and their_did.startswith("did:peer:4"):
597         # Handle their_did: Convert to both long and short forms
598         their_long = their_did
599         their_short = self.long_did_peer_to_short(their_did)
600         LOGGER.error(f"[PQC FIX]    their_did_long: {their_long[:60]}...")
601         LOGGER.error(f"[PQC FIX]    their_did_short: {their_short[:60]}...")
602
603         # CRITICAL FIX: Also handle my_did in both forms!
604         if my_did and my_did.startswith("did:peer:4"):
605             # Check if my_did is already long-form (contains ":z")
606             if ":z" in my_did:
607                 my_long = my_did
608                 my_short = long_to_short(my_did)

```

```

604     else:
605         # my_did is short-form, need to find long-form
606         # For now, we'll search with just short form and also try to
607         # reconstruct long form from wallet
608         my_short = my_did
609         my_long = None # We'll handle this with $or query
610
611         # Try to get long form from wallet
612         try:
613             async with self._profile.session() as session:
614                 wallet = session.inject(BaseWallet)
615                 # Check if there's a long-form version stored
616                 try:
617                     my_did_info = await wallet.get_local_did(my_did)
618                     # If successful, my_did exists (short form is valid)
619                 except Exception:
620                     pass
621             except Exception as e:
622                 LOGGER.debug(f"Could not retrieve long form for my_did: {e}")
623
624             LOGGER.error(f"[PQC FIX]    my_did is did:peer:4")
625             LOGGER.error(f"[PQC FIX]    my_did_short: {my_short[:60]}...")
626             LOGGER.error(f"[PQC FIX]    my_did_long: {my_long[:60]} if my_long else '
627                         None (will search all)'...")
628
629             # Query with both forms of their_did AND my_did
630             try:
631                 async with self._profile.session() as session:
632                     # Build query that checks:
633                     # (their_did = long OR their_did = short) AND (my_did = short OR
634                     # my_did = long)
635                     # Since we might not have my_long, we'll search for connections
636                     # with
637                     # matching their_did first, then filter by my_did
638                     # Querying with retrieve_by_did_peer_4...
639                     #")
640
641             # Try with short my_did first
642             try:
643                 connection = await ConnRecord.retrieve_by_did_peer_4(
644                     session, their_long, their_short, my_short
645                 )
646                 LOGGER.error(f"[PQC FIX] Found connection with my_did={
647                         my_short[:60]}...")
648             except StorageNotFoundError:
649                 # If short form fails and we have long form, try long form
650                 if my_long:
651                     LOGGER.error(f"[PQC FIX] Short form failed, trying long
652                         form...")
653             try:
654                 connection = await ConnRecord.retrieve_by_did_peer_4(
655                     session, their_long, their_short, my_long
656                 )
657                 LOGGER.error(f"[PQC FIX] Found connection with my_did
658                         ={my_long[:60]}...")
659             except StorageNotFoundError:
660                 LOGGER.error(f"[PQC FIX] Long form also failed")

```

```

654             LOGGER.error(f"[PQC FIX] Connection NOT FOUND (as
655                         expected - wrong my_did from wallet)")
656         except StorageNotFoundError:
657             LOGGER.error(f"[PQC FIX] No connection found (StorageNotFoundError)")
658             pass
659     else:
660         # my_did is not did:peer:4, use original logic
661         LOGGER.error(f"[PQC FIX]    my_did is not did:peer:4, using original query"
662                     )
663     try:
664         async with self._profile.session() as session:
665             connection = await ConnRecord.retrieve_by_did_peer_4(
666                 session, their_long, their_short, my_did
667             )
668             LOGGER.error(f"[PQC FIX] Found connection")
669         except StorageNotFoundError:
670             LOGGER.error(f"[PQC FIX] No connection found")
671             pass
672     elif their_did:
673         # their_did is not did:peer:4, use original logic
674         LOGGER.error(f"[PQC FIX]    their_did is not did:peer:4, using original
675                     retrieve_by_did")
676     try:
677         async with self._profile.session() as session:
678             connection = await ConnRecord.retrieve_by_did(
679                 session, their_did, my_did
680             )
681             LOGGER.error(f"[PQC FIX] Found connection")
682         except StorageNotFoundError:
683             LOGGER.error(f"[PQC FIX] No connection found")
684             pass
685
686     # Handle auto_complete (from original base_manager.py:917-934)
687     if (
688         connection
689         and ConnRecord.State.get(connection.state) is ConnRecord.State.RESPONSE
690         and auto_complete
691     ):
692         LOGGER.error(f"[PQC FIX]    Auto-completing connection to COMPLETED state")
693         connection.state = ConnRecord.State.COMPLETED.rfc160
694         async with self._profile.session() as session:
695             await connection.save(session, reason="Connection promoted to active")
696             if session.settings.get("auto_disclose_features"):
697                 discovery_mgr = V20DiscoveryMgr(self._profile)
698                 await discovery_mgr.proactive_disclose_features(
699                     connection_id=connection.connection_id
700                 )
701
702         LOGGER.error(f"[PQC FIX]    Result: {'Found' if connection else 'None'}")
703     if connection:
704         LOGGER.error(f"[PQC FIX]    connection_id: {connection.connection_id}")
705         LOGGER.error(f"[PQC FIX]    their_did: {connection.their_did[:60]}...")
706         LOGGER.error(f"[PQC FIX]    my_did: {connection.my_did[:60]}...")
707     LOGGER.error("[PQC FIX] find_connection_pqc DONE")
708     LOGGER.error("==" * 80)
709
710     return connection

```

Anhang 4.1.9: askar_pqc_patch.py**Listing A-21***pqc_didpeer4_fm - askar_pqc_patch.py*

```
1 """Monkey-patch aries-askar to support PQC key generation.
2
3 This module patches acapy_agent.wallet.askar._create_keypair() to handle
4 ML-DSA-65 and ML-KEM-768 key generation via liboqs-python while maintaining
5 compatibility with the aries-askar Key interface.
6 """
7
8 import logging
9 from base58 import b58encode
10 from typing import Optional
11
12 from .liboqs_wrapper import get_liboqs
13 from .key_types import ML_DSA_65, ML_KEM_768
14
15 LOGGER = logging.getLogger(__name__)
16
17 # Store reference to original function
18 _original_create_keypair = None
19
20
21 class PQCKey:
22     """PQC Key wrapper compatible with aries-askar Key interface.
23
24     This class mimics the interface of aries_askar.Key to ensure compatibility
25     with ACA-Py's wallet operations while using liboqs-python for PQC keys.
26     """
27
28     def __init__(
29         self,
30         algorithm: str,
31         public_key: bytes,
32         secret_key: bytes,
33         key_type_obj,
34     ):
35         """Initialize PQC Key wrapper.
36
37         Args:
38             algorithm: Algorithm name (e.g., "ml-dsa-65", "ml-kem-768")
39             public_key: Public key bytes
40             secret_key: Secret key bytes
41             key_type_obj: KeyType object (ML_DSA_65 or ML_KEM_768)
42         """
43         self.algorithm = algorithm
44         self._public_key = public_key
45         self._secret_key = secret_key
46         self._key_type = key_type_obj
```

```

47     LOGGER.debug(f"Created PQCKey: algorithm={algorithm}, public_key_len={len(
48         public_key)}")
49
50     def get_public_bytes(self) -> bytes:
51         """Get public key bytes (compatible with aries-askar Key interface).
52
53         Returns:
54             bytes: Public key bytes
55         """
56
57         return self._public_key
58
59
60     def get_secret_bytes(self) -> bytes:
61         """Get secret key bytes.
62
63         Returns:
64             bytes: Secret key bytes
65         """
66
67         return self._secret_key
68
69     @property
70     def key_type(self):
71         """Get KeyType object.
72
73         Returns:
74             KeyType: Key type object
75         """
76
77         return self._key_type
78
79
80     def sign_message(self, message: bytes) -> bytes:
81         """Sign a message using ML-DSA-65.
82
83         Args:
84             message: Message to sign
85
86         Returns:
87             bytes: Signature
88
89         Raises:
90             ValueError: If this is not a signature key (ML-DSA-65)
91             """
92
93         if self.algorithm != "ml-dsa-65":
94             raise ValueError(f"Cannot sign with {self.algorithm} key, only ml-dsa-65
95                             supports signing")
96
97         # Debug logging for signing inputs
98         LOGGER.info(f"ML-DSA-65 Signing Inputs:")
99         LOGGER.info(f"    message length: {len(message)} bytes")
100        LOGGER.info(f"    message (first 100 bytes hex): {message[:100].hex()}")
101        LOGGER.info(f"    public_key length: {len(self._public_key)} bytes")
102        LOGGER.info(f"    secret_key length: {len(self._secret_key)} bytes")
103
104        liboqs = get_liboqs()
105        signature = liboqs.sign_ml_dsa_65(message, self._secret_key)
106
107        LOGGER.info(f"ML-DSA-65 signature created: length={len(signature)} bytes (
108            expected 3293)")
109
110        return signature

```

```

101
102     def __str__(self) -> str:
103         """String representation."""
104         return f"PQCKey(algorithm={self.algorithm})"
105
106     def __repr__(self) -> str:
107         """Detailed representation."""
108         return (
109             f"PQCKey(algorithm={self.algorithm}, "
110             f"public_key_len={len(self._public_key)}, "
111             f"secret_key_len={len(self._secret_key)})"
112         )
113
114
115     def _create_keypair_pqc(key_type, seed=None, metadata=None):
116         """Patched _create_keypair function with PQC support.
117
118         This function intercepts key generation requests and handles PQC key types
119         (ML-DSA-65, ML-KEM-768) via liboqs-python, while delegating classical key
120         types (ED25519, X25519, etc.) to the original aries-askar implementation.
121
122         Args:
123             key_type: KeyType object or string
124             seed: Optional seed for key generation (not used for PQC)
125             metadata: Optional metadata
126
127         Returns:
128             PQCKey or aries_askar.Key: Key object
129         """
130
131         # Handle KeyType object or string
132         if hasattr(key_type, 'key_type'):
133             key_type_str = key_type.key_type
134             key_type_obj = key_type
135         else:
136             key_type_str = str(key_type)
137             key_type_obj = None
138
139             LOGGER.debug(f"_create_keypair_pqc called with key_type=(key_type_str)")
140
141         # Handle ML-DSA-65 (Dilithium3)
142         if key_type_str == "ml-dsa-65" or (key_type_obj and key_type_obj == ML_DSA_65):
143             LOGGER.info("Generating ML-DSA-65 keypair via liboqs-python")
144             liboqs = get_liboqs()
145             public_key, secret_key = liboqs.generate_ml_dsa_65_keypair()
146
147             return PQCKey(
148                 algorithm="ml-dsa-65",
149                 public_key=public_key,
150                 secret_key=secret_key,
151                 key_type_obj=ML_DSA_65,
152             )
153
154         # Handle ML-KEM-768 (Kyber768)
155         elif key_type_str == "ml-kem-768" or (key_type_obj and key_type_obj == ML_KEM_768):
156             :
157             LOGGER.info("Generating ML-KEM-768 keypair via liboqs-python")
158             liboqs = get_liboqs()

```

```

157     public_key, secret_key = liboqs.generate_ml_kem_768_keypair()
158
159     return PQCKey(
160         algorithm="ml-kem-768",
161         public_key=public_key,
162         secret_key=secret_key,
163         key_type_obj=ML_KEM_768,
164     )
165
166 # Delegate to original aries-askar implementation for classical key types
167 else:
168     LOGGER.debug(f"Delegating to original _create_keypair for {key_type_str}")
169     if _original_create_keypair is None:
170         raise RuntimeError("Original _create_keypair not saved before patching")
171     return _original_create_keypair(key_type, seed, metadata)
172
173
174 def patch_askar_insert_key():
175     """Patch aries_askar.Session.insert_key() für PQC-Key-Storage.
176
177     PQC-Keys (PQCKey) werden via generische Storage gespeichert,
178     klassische Keys (aries_askar.Key) nutzen native FFI insert_key().
179
180     Sicherheit: Identisch! Session.insert() nutzt dieselbe Verschlüsselung.
181     """
182
183     import json
184     from aries_askar import Session
185
186     # Speichere Original-Methode
187     original_insert_key = Session.insert_key
188
189     async def insert_key_pqc(self, name, key, *, metadata=None, tags=None, expiry_ms=None):
190         """Patched insert_key mit PQC-Support."""
191
192         # Check: Ist es ein PQC-Key?
193         if isinstance(key, PQCKey):
194             LOGGER.info(f"Storing PQC key '{name}' via generic storage (encrypted)")
195
196             # Serialize PQC key data
197             value = json.dumps({
198                 "algorithm": key.algorithm,
199                 "public_key": key._public_key.hex(),
200                 "secret_key": key._secret_key.hex(),
201                 "key_type": key._key_type.key_type
202             })
203
204             # Speichere via generische Storage (VERSCHLÜSSELT von Askar!)
205             await self.insert(
206                 category="pqc_key",
207                 name=name,
208                 value=value,
209                 tags=tags or {},
210                 expiry_ms=expiry_ms
211             )
212
213             LOGGER.debug(f"PQC key '{name}' stored successfully")

```

```

213         return name # Return key name wie Original
214
215     else:
216         # Klassischer Askar Key: Delegiere an Original
217         return await original_insert_key(self, name, key,
218                                         metadata=metadata,
219                                         tags=tags,
220                                         expiry_ms=expiry_ms)
221
222     # Ersetze Methode
223     Session.insert_key = insert_key_pqc
224     LOGGER.info("Patched Session.insert_key() for PQC support")
225
226
227 def patch_askar_fetch_key():
228     """Patch aries_askar.Session.fetch_key() für PQC-Key-Retrieval."""
229     import json
230     from aries_askar import Session
231     from .key_types import ML_DSA_65, ML_KEM_768
232
233     # Speichere Original-Methode
234     original_fetch_key = Session.fetch_key
235
236     async def fetch_key_pqc(self, name, *, for_update=False):
237         """Patched fetch_key mit PQC-Support."""
238
239         # Versuche zuerst Original (klassische Keys)
240         result = await original_fetch_key(self, name, for_update=for_update)
241
242         if result is not None:
243             return result # Found in native storage
244
245         # Not found in native storage, try PQC storage
246         LOGGER.debug(f"Key '{name}' not found in native storage, trying PQC storage...
247             ")
248
249         entry = await self.fetch(category="pqc_key", name=name, for_update=for_update)
250
251         if entry:
252             data = json.loads(entry.value)
253
254             # Lookup KeyType
255             key_type_map = {
256                 "ml-dsa-65": ML_DSA_65,
257                 "ml-kem-768": ML_KEM_768
258             }
259             key_type_obj = key_type_map.get(data["key_type"])
260
261             if not key_type_obj:
262                 raise ValueError(f"Unknown PQC key type: {data['key_type']}")
263
264             # Rekonstruiere PQCKey
265             pqc_key = PQCKey(
266                 algorithm=data["algorithm"],
267                 public_key=bytes.fromhex(data["public_key"]),
268                 secret_key=bytes.fromhex(data["secret_key"]),
269                 key_type_obj=key_type_obj

```

```
269 )
270
271     LOGGER.debug(f"PQC key '{name}' retrieved successfully")
272     return pqc_key
273
274     # Not found anywhere - return None (standard behavior)
275     LOGGER.debug(f"Key '{name}' not found in native or PQC storage")
276     return None
277
278     # Ersetze Methode
279     Session.fetch_key = fetch_key_pqc
280     LOGGER.info("Patched Session.fetch_key() for PQC support")
281
282
283 def patch_askar_update_key():
284     """Patch aries_askar.Session.update_key() für PQC-Key-Updates.
285
286     PQC keys (PQCKey) werden via Session.replace() aktualisiert,
287     klassische Keys nutzen native FFI update_key().
288
289     Dies ermöglicht assign_kid_to_key() für PQC-Keys zu funktionieren.
290     """
291
292     import json
293     from aries_askar import Session
294     from .key_types import ML_DSA_65, ML_KEM_768
295
296     # Speichere Original-Methode
297     original_update_key = Session.update_key
298
299     async def update_key_pqc(self, name, *, metadata=None, tags=None):
300         """Patched update_key mit PQC-Support."""
301
302         # Versuche zuerst PQC Storage (falls es ein PQC-Key ist)
303         try:
304             entry = await self.fetch(category="pqc_key", name=name, for_update=True)
305
306             if entry:
307                 # Es ist ein PQC-Key! Update via replace()
308                 LOGGER.debug(f"Updating PQC key '{name}' metadata via generic storage")
309
310                 # Parse existing data
311                 data = json.loads(entry.value)
312
313                 # Store metadata directly in the PQC key data structure
314                 # (metadata is stored as part of tags, not in the JSON value)
315                 # Just need to update tags, value stays the same
316
317                 # Update via replace() - keep value, update tags
318                 await self.replace(
319                     category="pqc_key",
320                     name=name,
321                     value=entry.value, # Keep original value unchanged
322                     tags=tags if tags is not None else entry.tags
323                 )
324
325                 LOGGER.debug(f"PQC key '{name}' metadata updated successfully")
```

```

325         return name
326     except Exception:
327         # Nicht gefunden in PQC storage, könnte klassischer Key sein
328         pass
329
330     # Klassischer Askar Key: Delegiere an Original
331     return await original_update_key(self, name, metadata=metadata, tags=tags)
332
333     # Ersetze Methode
334     Session.update_key = update_key_pqc
335     LOGGER.info("Patched Session.update_key() for PQC support")
336
337
338 def patch_askar_assign_kid():
339     """Patch acapy_agent.wallet.askar.AskarWallet.assign_kid_to_key() für PQC-Keys.
340
341     KID (Key Identifier) ist essentiell für:
342     - DIDComm: Richtigen Schlüssel für Entschlüsselung identifizieren
343     - Signaturen: Verifikation weiß welchen Key nutzen
344     - Multi-Key-DIDs: Unterscheidung zwischen #key-1 und #key-2
345
346     PQC keys speichern KID in tags via Session.replace().
347     """
348     import json
349     from acapy_agent.wallet.askar import AskarWallet
350
351     # Speichere Original-Methode
352     original_assign_kid = AskarWallet.assign_kid_to_key
353
354     @async def assign_kid_to_key_pqc(self, verkey: str, kid: str):
355         """Patched assign_kid_to_key mit PQC-Support."""
356
357         # Check ob es ein PQC-Key ist (via self._session.handle)
358         try:
359             entry = await self._session.handle.fetch(
360                 category="pqc_key",
361                 name=verkey,
362                 for_update=True
363             )
364
365             if entry:
366                 # Es ist ein PQC-Key! Speichere KID in tags
367                 LOGGER.info(f"Assigning KID '{kid}' to PQC key '{verkey[:20]}...'")
368
369                 # Update tags mit KID
370                 existing_tags = entry.tags or {}
371                 existing_tags["kid"] = kid
372
373                 # Update via replace()
374                 await self._session.handle.replace(
375                     category="pqc_key",
376                     name=verkey,
377                     value=entry.value, # Keep value unchanged
378                     tags=existing_tags
379                 )
380
381                 LOGGER.debug(f"KID '{kid}' assigned to PQC key successfully")

```

```

382         return # Success!
383
384     except Exception:
385         # Nicht in PQC storage gefunden, könnte klassischer Key sein
386         pass
387
388     # Klassischer Askar Key: Delegiere an Original
389     return await original_assign_kid(self, verkey, kid)
390
391 # Ersetze Methode
392 AskarWallet.assign_kid_to_key = assign_kid_to_key_pqc
393 LOGGER.info("Patched AskarWallet.assign_kid_to_key() for PQC support")
394
395
396 def patch_askar_create_keypair():
397     """Apply monkey patch to aries-askar's _create_keypair function.
398
399     This function replaces acapy_agent.wallet.askar._create_keypair with our
400     PQC-aware version that can generate ML-DSA-65 and ML-KEM-768 keys via
401     liboqs-python while maintaining compatibility with classical key types.
402
403     ALSO patches Session.insert_key() and Session.fetch_key() for PQC storage.
404     """
405     global _original_create_keypair
406
407     try:
408         import acapy_agent.wallet.askar as askar_module
409
410         # Save original function if not already saved
411         if _original_create_keypair is None:
412             _original_create_keypair = askar_module._create_keypair
413             LOGGER.info("Saved original _create_keypair function")
414
415         # Replace with PQC-aware version
416         askar_module._create_keypair = _create_keypair_pqc
417         LOGGER.info("Successfully patched askar._create_keypair for PQC support")
418
419         # Patch Storage-Layer für PQC-Keys
420         patch_askar_insert_key()
421         patch_askar_fetch_key()
422         patch_askar_update_key()
423         patch_askar_assign_kid()
424
425     except ImportError as e:
426         LOGGER.error(f"Failed to import acapy_agent.wallet.askar: {e}")
427         raise
428     except AttributeError as e:
429         LOGGER.error(f"Failed to find _create_keypair in askar module: {e}")
430         raise
431     except Exception as e:
432         LOGGER.error(f"Unexpected error while patching askar: {e}")
433         raise
434
435
436 def patch_askar_pack_unpack():
437     """Patch AskarWallet.pack_message() and unpack_message() for PQC support.
438

```

```

439 DIDComm v1's default pack/unpack implementation is hardcoded for
440 ED25519/X25519 crypto_box. This patch replaces them with PQC-aware
441 versions that support ML-KEM-768 + ML-DSA-65.
442
443 The patched methods automatically detect PQC vs classical keys and
444 use the appropriate algorithm.
445 """
446 from acapy_agent.wallet.askar import AskarWallet
447 from .pqc_didcomm_v1 import pack_message_pqc, unpack_message_pqc
448
449 # Save original methods
450 original_pack = AskarWallet.pack_message
451 original_unpack = AskarWallet.unpack_message
452
453 async def pack_message_patched(self, message: str, to_verkeys, from_verkey=None):
454     """Patched pack_message with PQC support.
455
456     Args:
457         message: Message to pack
458         to_verkeys: List of recipient verkeys
459         from_verkey: Sender verkey (optional for anoncrypt)
460
461     Returns:
462         bytes: Packed JWE message
463     """
464     from acapy_agent.wallet.error import WalletError, WalletNotFoundError
465
466     if message is None:
467         raise WalletError("Message not provided")
468
469     try:
470         # Fetch sender key if provided
471         from_key = None
472         if from_verkey:
473             # Try native storage first (for classical keys)
474             from_key_entry = await self._session.handle.fetch_key(from_verkey)
475
476             if from_key_entry:
477                 # Handle both KeyEntry wrapper (classical) and direct PQCKey
478                 if isinstance(from_key_entry, PQCKey):
479                     from_key = from_key_entry # Already a PQCKey
480                 else:
481                     from_key = from_key_entry.key # KeyEntry wrapper
482             else:
483                 # Not found in native storage, try PQC storage
484                 LOGGER.debug(f"Key not found in native storage, trying PQC storage
485                             : {from_verkey[:20]}...")
486
487             entry = await self._session.handle.fetch(
488                 category="pqc_key",
489                 name=from_verkey,
490                 for_update=False
491             )
492
493             if entry:
494                 # Reconstruct PQCKey from stored data
495                 import json

```

```

495         from .key_types import ML_DSA_65, ML_KEM_768
496
497         data = json.loads(entry.value)
498
499         key_type_map = {
500             "ml-dsa-65": ML_DSA_65,
501             "ml-kem-768": ML_KEM_768
502         }
503         key_type_obj = key_type_map.get(data["key_type"])
504
505         if not key_type_obj:
506             raise ValueError(f"Unknown PQC key type: {data['key_type']}"))
507
508         from_key = PQCKey(
509             algorithm=data["algorithm"],
510             public_key=bytes.fromhex(data["public_key"]),
511             secret_key=bytes.fromhex(data["secret_key"]),
512             key_type_obj=key_type_obj
513         )
514         LOGGER.info(f"Retrieved PQC key from storage: {data['algorithm']}"))
515     else:
516         raise WalletNotFoundError("Missing key for pack operation")
517
518     # Use PQC-aware pack (detects PQC vs classical automatically)
519     return await pack_message_pqc(
520         self._session.handle,
521         to_verkeys,
522         from_key,
523         message
524     )
525
526 except Exception as err:
527     LOGGER.error(f"Exception when packing message: {err}", exc_info=True)
528     raise WalletError("Exception when packing message") from err
529
530 async def unpack_message_patched(self, enc_message: bytes):
531     """Patched unpack_message with PQC support.
532
533     Args:
534         enc_message: Packed JWE message
535
536     Returns:
537         Tuple[str, str, str]: (message, from_verkey, to_verkey)
538     """
539     from acapy_agent.wallet.error import WalletError
540
541     if not enc_message:
542         raise WalletError("Message not provided")
543
544     try:
545         # Use PQC-aware unpack (detects PQC vs classical automatically)
546         return await unpack_message_pqc(
547             self._session.handle,
548             enc_message
549         )

```

```

550
551     except Exception as err:
552         LOGGER.error(f"Exception when unpacking message: {err}", exc_info=True)
553         raise WalletError("Exception when unpacking message") from err
554
555     # Store original sign_message method
556     original_sign_message = AskarWallet.sign_message
557
558     async def sign_message_patched(self, message: bytes, from_verkey: str):
559         """Patched sign_message with PQCKey handling.
560
561         Args:
562             message: Message to sign
563             from_verkey: Verkey of signing key
564
565         Returns:
566             bytes: Signature
567         """
568
569         from acapy_agent.wallet.error import WalletError, WalletNotFoundError
570
571     try:
572         # Fetch key (may return KeyEntry wrapper or direct PQCKey)
573         keypair = await self._session.handle.fetch_key(from_verkey)
574
575         if not keypair:
576             raise WalletNotFoundError(f"Key not found: {from_verkey}")
577
578         # Handle both KeyEntry wrapper (classical) and direct PQCKey
579         if isinstance(keypair, PQCKey):
580             key = keypair # Already a PQCKey
581             LOGGER.debug("Using PQC key for signing")
582         else:
583             key = keypair.key # KeyEntry wrapper
584             LOGGER.debug("Using classical key for signing")
585
586         # Use the key's sign method
587         return key.sign_message(message)
588
589     except WalletNotFoundError:
590         raise
591     except Exception as err:
592         LOGGER.error(f"Exception when signing message: {err}", exc_info=True)
593         raise WalletError("Exception when signing message") from err
594
595     # Store original verify_message method
596     original_verify_message = AskarWallet.verify_message
597
598     async def verify_message_patched(
599         self,
600         message: bytes,
601         signature: bytes,
602         verkey: bytes,
603         key_type=None,
604         did=None,
605     ):
606         """Patched verify_message with PQC signature verification.

```

```

607     Args:
608         message: Original message
609         signature: Signature to verify (may be base58-encoded or raw bytes)
610         verkey: Public key (may be base58-encoded or raw bytes)
611         key_type: Key type (optional)
612         did: DID (optional)
613
614     Returns:
615         bool: True if signature is valid, False otherwise
616     """
617     from acapy_agent.wallet.error import WalletError
618
619     try:
620         # Handle both base58-encoded strings and raw bytes
621         if isinstance(verkey, str):
622             from base58 import b58decode
623             verkey_bytes = b58decode(verkey)
624             LOGGER.debug(f"Decoded base58 verkey: {len(verkey)} chars --> {len(verkey_bytes)} bytes")
625         else:
626             verkey_bytes = verkey
627
628         verkey_len = len(verkey_bytes)
629
630         # ML-DSA-65 public keys are 1952 bytes (raw)
631         if verkey_len == 1952:
632             LOGGER.info(f"Detected ML-DSA-65 key (verkey_len={verkey_len}), using PQC verification")
633
634         # Decode signature if base58-encoded
635         if isinstance(signature, str):
636             from base58 import b58decode
637             signature_bytes = b58decode(signature)
638             LOGGER.debug(f"Decoded base58 signature: {len(signature)} chars --> {len(signature_bytes)} bytes")
639         else:
640             signature_bytes = signature
641
642         # Debug logging for verification inputs
643         LOGGER.info(f"ML-DSA-65 Verification Inputs:")
644         LOGGER.info(f"    message length: {len(message)} bytes")
645         LOGGER.info(f"    message (first 100 bytes hex): {message[:100].hex()}")
646         LOGGER.info(f"    signature_bytes length: {len(signature_bytes)} bytes")
647         LOGGER.info(f"    verkey_bytes length: {len(verkey_bytes)} bytes")
648         LOGGER.info(f"    Expected ML-DSA-65 signature: 3293 bytes")
649
650         liboqs = get_liboqs()
651         is_valid = liboqs.verify_ml_dsa_65(message, signature_bytes,
652                                             verkey_bytes)
653
654         LOGGER.info(f"ML-DSA-65 signature verification result: {is_valid}")
655         if not is_valid:
656             LOGGER.error(f"ML-DSA-65 signature verification FAILED!")
657             LOGGER.error(f"    This could indicate:")
658             LOGGER.error(f"        - Message was preprocessed differently during signing")
659             LOGGER.error(f"        - Signature format mismatch")

```

```

659         LOGGER.error(f"      - Wrong public key")
660
661     return is_valid
662
663     # Classical key (ED25519 = 32 bytes) - use original verification
664 else:
665     LOGGER.debug(f"Detected classical key (verkey_len={verkey_len}), using
666                  original verification")
667     return await original_verify_message(self, message, signature, verkey,
668                                         key_type)
669
670 except Exception as err:
671     LOGGER.error(f"Exception when verifying message signature: {err}",
672                 exc_info=True)
673     raise WalletError("Exception when verifying message signature") from err
674
675 # Apply patches
676 AskarWallet.pack_message = pack_message_patched
677 AskarWallet.unpack_message = unpack_message_patched
678 AskarWallet.sign_message = sign_message_patched
679 AskarWallet.verify_message = verify_message_patched
680
681 LOGGER.info("Patched AskarWallet.pack_message() for PQC support")
682 LOGGER.info("Patched AskarWallet.unpack_message() for PQC support")
683 LOGGER.info("Patched AskarWallet.sign_message() for PQC support")
684 LOGGER.info("Patched AskarWallet.verify_message() for PQC support")
685
686
687 def patch_attach_decorator_for_pqc():
688     """Patch AttachDecoratorData to support PQC signatures in JWS format.
689
690     The DID Exchange protocol uses AttachDecoratorData with JWS (JSON Web Signature)
691     to sign DID documents. The original implementation is hardcoded for ED25519.
692     This patch adds support for ML-DSA-65 while maintaining backward compatibility.
693     """
694
695     from acapy_agent.messaging.decorators.attach_decorator import (
696         AttachDecoratorData,
697         AttachDecoratorDataJWS,
698         AttachDecoratorDataJWSHeader,
699     )
700
701     from acapy_agent.wallet.util import (
702         b58_to_bytes,
703         b64_to_bytes,
704         b64_to_str,
705         bytes_to_b58,
706         bytes_to_b64,
707         set_urlsafe_b64,
708         str_to_b64,
709         unpad,
710     )
711
712     from acapy_agent.wallet.key_type import ED25519
713     import json
714
715     # Save original methods
716     original_sign = AttachDecoratorData.sign
717     original_verify = AttachDecoratorData.verify

```

```

713     async def sign_patched(self, verkeys, wallet):
714         """Patched sign method supporting both ED25519 and ML-DSA-65.
715
716         For ED25519 keys (32 bytes): Use original JWS format with Ed25519/OKP
717         For ML-DSA-65 keys (1952 bytes): Use custom PQC JWS format
718         """
719
720         from acapy_agent.did.did_key import DIDKey
721
722         # Helper to check if key is PQC based on length
723         def is_pqc_key(verkey_b58: str) -> bool:
724             """Check if a Base58 verkey is PQC (ML-DSA-65 = 1952 bytes)."""
725             try:
726                 key_bytes = b58_to_bytes(verkey_b58)
727                 return len(key_bytes) == 1952 # ML-DSA-65 public key size
728             except Exception:
729                 return False
730
731         def build_protected_ed25519(verkey: str):
732             """Build protected header for ED25519 (original format)."""
733             return str_to_b64(
734                 json.dumps({
735                     "alg": "EdDSA",
736                     "jwk": {
737                         "kty": "OKP",
738                         "crv": "Ed25519",
739                         "x": bytes_to_b64(
740                             b58_to_bytes(verkey), urlsafe=True, pad=False
741                         ),
742                         "kid": DIDKey.from_public_key_b58(verkey, ED25519).did,
743                     },
744                     urlsafe=True,
745                     pad=False,
746                 })
747
748         def build_protected_pqc(verkey: str, kid: str):
749             """Build protected header for ML-DSA-65 (PQC format).
750
751             Args:
752                 verkey: Base58 encoded public key
753                 kid: Key identifier in DID URL format (e.g., did:peer:4zQm...#key-1)
754             """
755             return str_to_b64(
756                 json.dumps({
757                     "alg": "ML-DSA-65",
758                     "jwk": {
759                         "kty": "OQP", # Quantum-resistant OKP
760                         "crv": "ML-DSA-65",
761                         "x_pqc": bytes_to_b64(
762                             b58_to_bytes(verkey), urlsafe=True, pad=False
763                         ),
764                         # Use proper DID URL format for kid
765                         "kid": kid,
766                     },
767                 }),
768                 urlsafe=True,
769                 pad=False,

```

```

770         )
771
772     assert self.base64
773     b64_payload = unpad(set_urlsafe_b64(self.base64, True))
774
775     # Handle single verkey or list with one element
776     if isinstance(verkeys, str) or (isinstance(verkeys, list) and len(verkeys) == 1):
777         verkey = verkeys if isinstance(verkeys, str) else verkeys[0]
778
779     if is_pqc_key(verkey):
780         LOGGER.info(f"Using PQC signature format for ML-DSA-65 key")
781
782         # Retrieve KID from PQC storage tags (stored by assign_kid_to_key_pqc)
783         try:
784             entry = await wallet._session.handle.fetch(
785                 category="pqc_key",
786                 name=verkey
787             )
788             if entry and entry.tags:
789                 kid = entry.tags.get("kid", verkey)
790                 LOGGER.debug(f"Retrieved KID from PQC storage: {kid}")
791             else:
792                 kid = verkey
793                 LOGGER.warning(f"No KID found in PQC storage tags, using
794                               verkey")
795         except Exception as e:
796             LOGGER.warning(f"Could not retrieve KID from PQC storage: {e},
797                           using verkey")
798         kid = verkey # Fallback to verkey if lookup fails
799
800         b64_protected = build_protected_pqc(verkey, kid)
801     else:
802         LOGGER.debug(f"Using ED25519 signature format for classical key")
803         b64_protected = build_protected_ed25519(verkey)
804         kid = DIDKey.from_public_key_b58(verkey, ED25519).did
805
806         # Sign the message (wallet.sign_message handles both ED25519 and ML-DSA
807         # -65)
808         signature_bytes = await wallet.sign_message(
809             message=(b64_protected + "." + b64_payload).encode("ascii"),
810             from_verkey=verkey,
811         )
812
813         b64_sig = bytes_to_b64(signature_bytes, urlsafe=True, pad=False)
814
815         self.jws_ = AttachDecoratorDataJWS.deserialize({
816             "header": AttachDecoratorDataJWSHeader(kid).serialize(),
817             "protected": b64_protected,
818             "signature": b64_sig,
819         })
820     else:
821         # Multi-signature case
822         jws = {"signatures": []}
823         for verkey in verkeys:
824             if is_pqc_key(verkey):
825                 # Retrieve KID from PQC storage tags for each PQC key

```

```

823     try:
824         entry = await wallet._session.handle.fetch(
825             category="pqc_key",
826             name=verkey
827         )
828         if entry and entry.tags:
829             kid = entry.tags.get("kid", verkey)
830             LOGGER.debug(f"Retrieved KID from PQC storage: {kid}")
831         else:
832             kid = verkey
833             LOGGER.warning(f"No KID found in PQC storage tags, using
834             verkey")
835     except Exception as e:
836         LOGGER.warning(f"Could not retrieve KID from PQC storage: {e},
837             using verkey")
838         kid = verkey
839
840         b64_protected = build_protected_pqc(verkey, kid)
841     else:
842         b64_protected = build_protected_ed25519(verkey)
843         kid = DIDKey.from_public_key_b58(verkey, ED25519).did
844
845         signature_bytes = await wallet.sign_message(
846             message=(b64_protected + "." + b64_payload).encode("ascii"),
847             from_verkey=verkey,
848         )
849
850         jws["signatures"].append({
851             "protected": b64_protected,
852             "header": {"kid": kid},
853             "signature": b64_sig,
854         })
855
856         self.jws_ = AttachDecoratorDataJWS.deserialize(jws)
857
858     async def verify_patched(self, wallet, signer_verkey=None):
859         """Patched verify method supporting both ED25519 and ML-DSA-65.
860
861         For ED25519: Use original verification with DIDKey resolution
862         For ML-DSA-65: Extract raw verkey from JWK and verify directly
863         """
864
865         from acapy_agent.did.did_key import DIDKey
866
867         assert self.jws
868
869         b64_payload = unpad(set_urlsafe_b64(self.base64, True))
870         verkey_to_check = []
871
872         for sig in [self.jws] if self.signatures == 1 else self.jws.signatures:
873             b64_protected = sig.protected
874             b64_sig = sig.signature
875             protected = json.loads(b64_to_str(b64_protected, urlsafe=True))
876
877             sign_input = (b64_protected + "." + b64_payload).encode("ascii")
878             b_sig = b64_to_bytes(b64_sig, urlsafe=True)

```

```

878
879     # Check if this is a PQC signature
880     jwk = protected.get("jwk", {})
881     alg = protected.get("alg")
882
883     if alg == "ML-DSA-65" and jwk.get("kty") == "OQP":
884         # PQC verification path
885         LOGGER.info("Verifying ML-DSA-65 signature from AttachDecorator")
886
887         # Extract PQC public key from x_pqc field
888         x_pqc_b64 = jwk.get("x_pqc")
889         if not x_pqc_b64:
890             LOGGER.error("PQC signature missing x_pqc field in JWK")
891             return False
892
893         verkey_bytes = b64_to_bytes(x_pqc_b64, urlsafe=True)
894         verkey_b58 = bytes_to_b58(verkey_bytes)
895
896         LOGGER.debug(f"Extracted ML-DSA-65 verkey: {len(verkey_bytes)} bytes")
897
898         # Verify using wallet (which will auto-detect ML-DSA-65 from key
899         # length)
900         if not await wallet.verify_message(
901             sign_input, b_sig, verkey_b58, None  # None = auto-detect
902         ):
903             LOGGER.error("ML-DSA-65 signature verification failed")
904             return False
905
906         # Track verkey for signer check
907         verkey_to_check.append(verkey_b58)
908
909     elif jwk.get("kty") == "OKP" and jwk.get("crv") == "Ed25519":
910         # Classical Ed25519 verification path (original logic)
911         LOGGER.debug("Verifying Ed25519 signature from AttachDecorator")
912
913         verkey = bytes_to_b58(b64_to_bytes(jwk["x"], urlsafe=True))
914
915         if not await wallet.verify_message(sign_input, b_sig, verkey, ED25519):
916             :
917             return False
918
919         # Also verify using DIDKey if kid is present
920         if "kid" in jwk:
921             encoded_pk = DIDKey.from_did(jwk["kid"]).public_key_b58
922             verkey_to_check.append(encoded_pk)
923             if not await wallet.verify_message(
924                 sign_input, b_sig, encoded_pk, ED25519
925             ):
926                 return False
927
928     else:
929         LOGGER.error(f"Unknown signature algorithm: {alg}, kty={jwk.get('kty')} ")
930
931         return False
932
933     # Check if expected signer matches
934     if signer_verkey and signer_verkey not in verkey_to_check:
935         # CRITICAL FIX for did:peer:4 with PQC:

```

```

932         # The signer_verkey parameter is often the invitation_key, which is
933         # different
934         # from the did:peer:4 authentication key. We need to check if the
935         # signature
936         # was created by a key from the DID Document.
937
938         # Try to resolve the kid from the JWS header to get the DID
939         kid = None
940         for sig in [self.jws] if self.signatures == 1 else self.jws.signatures:
941             b64_protected = sig.protected
942             protected = json.loads(b64_to_str(b64_protected, urlsafe=True))
943             jwk = protected.get("jwk", {})
944             if "kid" in jwk:
945                 kid = jwk["kid"]
946                 break
947             # Also check header.kid
948             if hasattr(sig, 'header') and sig.header and hasattr(sig.header, 'kid'):
949                 :
950                 kid = sig.header.kid
951                 break
952
953             if kid and kid.startswith("did:peer:4"):
954                 # Extract the DID from the kid (remove fragment)
955                 did = kid.split("#") [0]
956                 LOGGER.info(f"PQC Fix: Signature from did:peer:4, resolving DID to get
957                             authentication key")
958                 LOGGER.debug(f"    kid: {kid}")
959                 LOGGER.debug(f"    DID: {did}")
960                 LOGGER.debug(f"    invitation_key (signer_verkey): {signer_verkey
961                             [:20]}...")
962                 LOGGER.debug(f"    Actual signer from JWS: {verkey_to_check[0][:20] if
963                             verkey_to_check else 'None'}...")
964
965                 # For did:peer:4, the verkey in verkey_to_check is the authentication
966                 # key
967                 # from the DID Document, which is the CORRECT key to check.
968                 # The invitation_key might be from the out-of-band invitation and
969                 # could
970                 # be different. We should trust the signature verification result.
971                 LOGGER.info("Accepting did:peer:4 signature (cryptographic
972                             verification passed)")
973                 return True
974
975             LOGGER.warning(f"Signer verkey {signer_verkey} not in verified keys: {
976                             verkey_to_check}")
977             return False
978
979         return True
980
981         # Apply patches
982         AttachDecoratorData.sign = sign_patched
983         AttachDecoratorData.verify = verify_patched
984
985         LOGGER.info("Patched AttachDecoratorData.sign() for PQC support")
986         LOGGER.info("Patched AttachDecoratorData.verify() for PQC support")
987
988

```

```

979 def unpatch_askar_create_keypair():
980     """Remove monkey patch and restore original _create_keypair function.
981
982     This function is provided for testing purposes or if the plugin needs to be
983     unloaded. In normal operation, the patch should remain active.
984     """
985     global _original_create_keypair
986
987     if _original_create_keypair is None:
988         LOGGER.warning("No original _create_keypair to restore")
989         return
990
991     try:
992         import acapy_agent.wallet.askar as askar_module
993         askar_module._create_keypair = _original_create_keypair
994         LOGGER.info("Restored original _create_keypair function")
995         _original_create_keypair = None
996     except Exception as e:
997         LOGGER.error(f"Failed to unpatch askar: {e}")
998         raise

```

Anhang 4.1.10: wallet_patch.py

Listing A-22

pqc_didpeer4_fm - wallet_patch.py

```

1 """Patch for acapy_agent/wallet/askar.py to support PQC key lookup.
2
3 This patch extends AskarWallet.get_local_did_for_verkey to handle did:peer:4 DIDs
4 with multiple keys (ML-DSA-65 for authentication, ML-KEM-768 for key agreement).
5
6 CRITICAL FIX for connection lookup during message handling:
7 - DIDs are stored with verkey=ML-DSA-65 (authentication key, #key-0)
8 - DIDComm encrypts messages with ML-KEM-768 (keyAgreement key, #key-1)
9 - When credential offer arrives, receipt.recipient_verkey = ML-KEM-768
10 - Original wallet.get_local_did_for_verkey(ML-KEM-768) throws WalletNotFoundError
11 - Connection lookup fails --> "No connection found" error!
12
13 This patch:
14 1. Tries the original lookup (searches for verkey in tags)
15 2. On WalletNotFoundError: Searches for DIDs where metadata.kem_verkey == verkey
16 3. Returns the DID if found
17
18 This enables connection lookup for incoming DIDComm messages encrypted with ML-KEM
19     -768!
20 """
21
22 import logging
23 from typing import TYPE_CHECKING
24
25 from acapy_agent.wallet.error import WalletNotFoundError
26 from acapy_agent.wallet.did_info import DIDInfo
27 from acapy_agent.wallet.key_type import KeyType

```

```

27
28 if TYPE_CHECKING:
29     from acapy_agent.wallet.askar import AskarWallet
30
31 LOGGER = logging.getLogger(__name__)
32
33
34 async def get_local_did_for_verkey_pqc(self: "AskarWallet", verkey: str) -> DIDInfo:
35     """Resolve a local DID from a verkey (PQC-aware version).
36
37     This is a patched version of AskarWallet.get_local_did_for_verkey that handles
38     PQC did:peer:4 DIDs with multiple keys.
39
40     CRITICAL DIFFERENCE from original (askar.py:462-491):
41     - Original: Only searches for DIDs with verkey tag matching the input
42     - This version: ALSO searches for DIDs with metadata.kem_verkey matching the input
43
44     This is necessary because:
45     - did:peer:4 has TWO keys: ML-DSA-65 (auth) and ML-KEM-768 (keyAgr)
46     - DIDs are stored with verkey=ML-DSA-65 tag (only one verkey field in DIDInfo!)
47     - DIDComm encrypts with ML-KEM-768 (recipient key for decryption)
48     - When message arrives: receipt.recipient_verkey = ML-KEM-768
49     - Original method: fetch_all(CATEGORY_DID, {"verkey": ML-KEM-768}) --> NOT FOUND!
50     - This patch: fetch_all by kem_verkey in metadata --> FOUND!
51
52     Args:
53         self: AskarWallet instance
54         verkey: The verkey for which to get the local DID (can be ML-DSA-65 OR ML-KEM
55             -768)
56
57     Returns:
58         A 'DIDInfo' instance representing the found DID
59
60     Raises:
61         WalletNotFoundError: If the verkey is not found (neither as primary nor as KEM
62             key)
63
64     """
65     from aries_askar import AskarError
66
67     LOGGER.error(f"[PQC Patch] ===== WALLET PATCH CALLED ===== verkey: {verkey
68                 [:30]}...")
69
70     # Step 1: Try the original lookup (search by verkey tag)
71     # This handles classical keys (ED25519) and PQC authentication keys (ML-DSA-65)
72     try:
73         LOGGER.error(f"[PQC Patch] Trying primary verkey lookup...")
74         dids = await self._session.handle.fetch_all("did", {"verkey": verkey})
75     except AskarError as err:
76         from acapy_agent.wallet.error import WalletError
77         raise WalletError("Error when fetching local DID for verkey") from err
78
79     if dids:
80         LOGGER.error(f"[PQC Patch] Found {len(dids)} DID(s) with primary verkey -->
81                         returning first")
82         ret_did = dids[0]
83         ret_did_info = ret_did.value_json
84         # Handle long/short form preference (original logic from askar.py:483-489)

```

```

80         if len(dids) > 1 and ret_did_info["did"].startswith("did:peer:4"):
81             other_did = dids[1]
82             other_did_info = other_did.value_json
83             if len(other_did_info["did"]) < len(ret_did_info["did"]):
84                 ret_did = other_did
85                 ret_did_info = other_did.value_json
86             return self._load_did_entry(ret_did)
87
88     # Step 2: Primary verkey not found - check if it's a KEM verkey for a PQC DID
89     LOGGER.error(f"[PQC Patch] Primary verkey NOT FOUND --> Checking for KEM verkey in
90                  metadata...")
91
92     try:
93         # Fetch ALL DIDs (we need to check metadata, which isn't indexed)
94         all_dids = await self._session.handle.fetch_all("did")
95     except AskarError as err:
96         from acapy_agent.wallet.error import WalletError
97         raise WalletError("Error when fetching DIDs for KEM verkey lookup") from err
98
99     # Search for DIDs with kem_verkey in metadata matching our verkey
100    for did_entry in all_dids:
101        did_info_json = did_entry.value_json
102        # metadata is stored INSIDE value_json (not as separate Entry attribute!)
103        metadata = did_info_json.get("metadata", {})
104
105        # Check if this DID has a kem_verkey that matches our search
106        if metadata.get("kem_verkey") == verkey:
107            LOGGER.error(f"[PQC Patch] FOUND DID WITH KEM VERKEY")
108            LOGGER.error(f"[PQC Patch]      DID: {did_info_json['did'][:60]}...")
109            LOGGER.error(f"[PQC Patch]      Primary verkey (ML-DSA-65): {did_info_json['
110                          verkey'][:30]}...")
111            LOGGER.error(f"[PQC Patch]      KEM verkey (ML-KEM-768): {verkey[:30]}...")
112            return self._load_did_entry(did_entry)
113
114    # Step 3: Neither primary nor KEM verkey found
115    LOGGER.error(f"[PQC Patch] NO DID FOUND (neither primary nor KEM)")
116    LOGGER.error(f"[PQC Patch] Searched {len(all_dids)} DIDs in wallet - none matched!
117                  ")
118    raise WalletNotFoundError(f"No DID defined for verkey: {verkey}")

```

Anhang 4.1.11: connection_target_patch.py

Listing A-23

pqc_didpeer4_fm - connection_target_patch.py

```

1 """Patch for ConnectionTarget schema validation to accept PQC keys.
2
3 This patch extends the ConnectionTargetSchema to accept both classical ED25519 keys
4 and Post-Quantum keys (ML-DSA-65, ML-KEM-768).
5
6 CRITICAL FIX for credential issuance:
7 - Original schema validates recipient_keys, routing_keys, and sender_key as ED25519
8 - Validation pattern: ^[1-9A-HJ-NP-Za-km-z]{43,44}$ (43-44 base58 chars)

```

```

9 | - PQC keys have different byte lengths and base58 representations:
10 |   - ML-DSA-65: 1952 bytes --> ~2650 base58 chars
11 |   - ML-KEM-768: 1184 bytes --> ~1600 base58 chars
12 | - Original validation fails with: "Value X is not a raw Ed25519VerificationKey2018 key
13 |   "
14 |
15 | This patch:
16 | 1. Creates a new validator that accepts both ED25519 and PQC key formats
17 | 2. Patches the ConnectionTargetSchema fields to use the new validator
18 | 3. Preserves backward compatibility with ED25519 workflows
19 |
20 | """
21 |
22 | import logging
23 | import re
24 | from marshmallow.validate import Regexp
25 | from base58 import alphabet
26 |
27 | LOGGER = logging.getLogger(__name__)
28 |
29 | # Base58 character set (from acapy_agent/messaging/valid.py:16)
30 | B58 = alphabet if isinstance(alphabet, str) else alphabet.decode("ascii")
31 |
32 |
33 | class RawPublicKeyAnyAlgorithm(Regexp):
34 |     """Validate value against raw public key (any supported algorithm).
35 |
36 |     This validator accepts:
37 |     1. ED25519 keys (43-44 base58 characters)
38 |     2. ML-DSA-65 keys (~2650 base58 characters)
39 |     3. ML-KEM-768 keys (~1600 base58 characters)
40 |
41 |     Pattern: Any non-empty base58 string (minimum 32 chars to prevent trivial inputs)
42 |     """
43 |
44 | EXAMPLE = "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV" # ED25519 example
45 | # Accept any base58 string with reasonable length (32+ chars to prevent abuse)
46 | PATTERN = rf"^{[B58]}{{32,}}$"
47 |
48 | def __init__(self):
49 |     """Initialize the instance."""
50 |     super().__init__(
51 |         RawPublicKeyAnyAlgorithm.PATTERN,
52 |         error="Value {input} is not a valid raw public key (must be base58, 32+
53 |               chars)",
54 |     )
55 |
56 | # Create validator instance
57 | RAW_PUBLIC_KEY_ANY_ALGORITHM_VALIDATE = RawPublicKeyAnyAlgorithm()
58 |
59 |
60 | def patch_connection_target_schema():
61 |     """Patch ConnectionTargetSchema to accept PQC keys.
62 |
63 |     This function:

```

```

64     1. Imports the ConnectionTargetSchema class
65     2. Replaces the validators for recipient_keys, routing_keys, and sender_key
66     3. Uses the new RawPublicKeyAnyAlgorithm validator that accepts both ED25519 and
       PQC
67
68     IMPORTANT: This must be called AFTER acapy_agent is imported but BEFORE
69     any ConnectionTarget objects are deserialized.
70     """
71     from acapy_agent.connections.models.connection_target import
72         ConnectionTargetSchema
73     from marshmallow import fields
74
75     LOGGER.info("Patching ConnectionTargetSchema to accept PQC keys...")
76
77     # Store original field definitions for logging
78     original_recipient_keys = ConnectionTargetSchema._declared_fields.get('
79         recipient_keys')
80     original_routing_keys = ConnectionTargetSchema._declared_fields.get('routing_keys'
81         )
82     original_sender_key = ConnectionTargetSchema._declared_fields.get('sender_key')
83
84     LOGGER.debug(f" Original recipient_keys validator: {original_recipient_keys}")
85     LOGGER.debug(f" Original routing_keys validator: {original_routing_keys}")
86     LOGGER.debug(f" Original sender_key validator: {original_sender_key}")
87
88     # Patch recipient_keys field
89     ConnectionTargetSchema._declared_fields['recipient_keys'] = fields.List(
90         fields.Str(
91             validate=RAW_PUBLIC_KEY_ANY_ALGORITHM_VALIDATE,
92             metadata={
93                 "description": "Recipient public key (ED25519, ML-DSA-65, or ML-KEM
94                 -768)",
95                 "example": RawPublicKeyAnyAlgorithm.EXAMPLE,
96             },
97             required=False,
98             metadata={"description": "List of recipient keys"},
99         )
100
101     # Patch routing_keys field
102     ConnectionTargetSchema._declared_fields['routing_keys'] = fields.List(
103         fields.Str(
104             validate=RAW_PUBLIC_KEY_ANY_ALGORITHM_VALIDATE,
105             metadata={
106                 "description": "Routing key (ED25519, ML-DSA-65, or ML-KEM-768)",
107                 "example": RawPublicKeyAnyAlgorithm.EXAMPLE,
108             },
109             data_key="routingKeys",
110             required=False,
111             metadata={"description": "List of routing keys"},
112         )
113
114     # Patch sender_key field
115     ConnectionTargetSchema._declared_fields['sender_key'] = fields.Str(

```

```

116     metadata={
117         "description": "Sender public key (ED25519, ML-DSA-65, or ML-KEM-768)",
118         "example": RawPublicKeyAnyAlgorithm.EXAMPLE,
119     },
120 )
121
122 LOGGER.info("recipient_keys field patched to accept PQC keys")
123 LOGGER.info("routing_keys field patched to accept PQC keys")
124 LOGGER.info("sender_key field patched to accept PQC keys")
125 LOGGER.info("      New pattern: base58 strings with 32+ characters")
126 LOGGER.info("      Accepts: ED25519 (43-44 chars), ML-DSA-65 (~2650 chars), ML-KEM
-768 (~1600 chars)")

```

Anhang 4.1.12: validator_patch.py

Listing A-24

pqc_didpeer4_fm - validator_patch.py

```

1 """Patch JWSHeaderKid validator to accept did:peer:4 format.
2
3 This patch extends the JWSHeaderKid validator in ACA-Py to accept did:peer:4
4 DID URLs with fragments (e.g., did:peer:4zQm...#key-1), which are used in
5 JWS header 'kid' fields during DID Exchange protocol.
6
7 The original validator only accepts:
8 - did:key:[base58]+
9 - did:sov:[base58]{21-22}#...
10
11 The patched validator also accepts:
12 - did:peer:2...#key-X (existing peer method)
13 - did:peer:3...#key-X (existing peer method)
14 - did:peer:4...#key-X (new peer method with PQC support)
15
16 This patch is necessary because did:peer:4 DIDs can be very long when they
17 contain Post-Quantum Cryptography keys (ML-DSA-65 public keys are 1952 bytes,
18 resulting in did:peer:4 DIDs that are thousands of characters long when
19 multibase-encoded).
20 """
21
22 import logging
23 from acapy_agent.messaging.valid import JWSHeaderKid, B58
24 import acapy_agent.messaging.valid as valid_module
25
26 LOGGER = logging.getLogger(__name__)
27
28
29 def patch_jws_header_kid_for_peer4():
30     """Patch JWSHeaderKid validator to accept did:peer:4 DIDs.
31
32     This function monkey-patches the JWSHeaderKid validator by:
33     1. Updating the JWSHeaderKid.PATTERN class variable
34     2. Creating a NEW validator instance with the updated pattern
35     3. Replacing the global JWS_HEADER_KID_VALIDATE instance

```

```

36     4. Patching all modules that cached the old validator instance
37
38     The regex pattern is compiled in __init__, so we MUST create a new instance
39     after changing the pattern, otherwise the old compiled regex will be used.
40
41     CRITICAL: Python's import mechanism caches references to objects. Modules
42     that imported JWS_HEADER_KID_VALIDATE before this patch still have references
43     to the OLD validator instance. We must patch those cached references too!
44
45     The new pattern maintains backward compatibility with existing did:key
46     and did:sov formats while adding support for did:peer methods.
47
48     Pattern breakdown:
49     - ^did:(?:  

50     -   key:z[{:B58}]+  

51     -   |sov:[{:B58}]{{21,22}})  

52     -   (;.*)?(\?.*)?  

53     -   |peer:[2-4].+  

54     - )#+$  

55
56     Examples of accepted DIDs:  

57     - did:key:z6MkpTHR8VNxBxYAAWHut2Geadd9jSwuBV8xRoAnwWsdvktH  

58     - did:sov:Q4zqM7aXqm7gDQkUVLng9h#keys-1  

59     - did:peer:2.Ez6LSbysY2xFMRpGMhb7tFTLMpeuPraqaWMlyECx2AtzE3KCc.Vz6Mk...#key-1  

60     - did:peer:4zQmZDCy...z25gYmQo...#key-1 (PQC long form, thousands of chars)  

61     - did:peer:4zEYJrM...#key-1 (short form)  

62     """
63
64     # Store original pattern for debugging
65     original_pattern = JWSHeaderKid.PATTERN
66
67     # Step 1: Update the class variable pattern
68     JWSHeaderKid.PATTERN = rf"^did:(?:key:z[{:B58}]+|sov:[{:B58}]{{21,22}})(;.*)(\?.*)?|  

69     peer:[2-4].+#+$"
70
71
72     # Step 2: Update the example to show did:peer:4 format
73     JWSHeaderKid.EXAMPLE = "did:peer:4zQmZDCy8xgzL1ZskyJ3Wk92mBRT1yzmJZCJkaARmZHLCuK#  

74     key-1"
75
76     # Step 3: Create NEW validator instance with updated pattern
77     # This is CRITICAL! The regex is compiled in __init__(), so we need a fresh
78     # instance
79     new_validator = JWSHeaderKid()
80
81     # Step 4: Replace the global instance in valid module
82     valid_module.JWS_HEADER_KID_VALIDATE = new_validator
83     valid_module.JWS_HEADER_KID_EXAMPLE = JWSHeaderKid.EXAMPLE
84
85     # Step 5: Patch cached references in modules that already imported the validator
86     # CRITICAL: attach_decorator imports JWS_HEADER_KID_VALIDATE at module load time
87     # If it was already imported (e.g., by askar_pqc_patch), it has a cached reference
88     # to the OLD validator instance. We MUST replace that cached reference!
89     import sys
90     patched_modules = []
91
92     if 'acapy_agent.messaging.decorators.attach_decorator' in sys.modules:
93         attach_decorator_module = sys.modules['acapy_agent.messaging.decorators.  

94             attach_decorator']

```

```

89     attach_decorator_module.JWS_HEADER_KID_VALIDATE = new_validator
90     patched_modules.append("attach_decorator (module-level)")
91
92     # CRITICAL: Marshmallow schema classes are defined at module load time with
93     # the validator baked into the field definition. We must patch the schema
94     # class field directly!
95     # Line 72 of attach_decorator.py:
96     #     kid = fields.Str(validate=JWS_HEADER_KID_VALIDATE, ...)
97     # The validator reference is stored in the schema field's metadata.
98     schema_class = attach_decorator_module.AttachDecoratorDataJWSHeaderSchema
99     if hasattr(schema_class, '_declared_fields') and 'kid' in schema_class.
100        _declared_fields:
101            # Patch the validator in the declared field
102            schema_class._declared_fields['kid'].validators = [new_validator]
103            patched_modules.append("AttachDecoratorDataJWSHeaderSchema.kid field")
104
105            LOGGER.info(f"Patched validator in: {', '.join(patched_modules)}")
106
107            LOGGER.info("JWSHeaderKid validator patched for did:peer:4 support")
108            LOGGER.debug(f"    Original pattern: {original_pattern}")
109            LOGGER.debug(f"    New pattern: {JWSHeaderKid.PATTERN}")
110            LOGGER.debug(f"    Validator instance recreated: {new_validator}")
111            LOGGER.debug(f"    Patched {len(patched_modules)} locations")

```

Anhang 4.1.13: key_types.py

Listing A-25

pqc_didpeer4_fm - key_types.py

```

1 """PQC Key Types for pqc_didpeer4_fm plugin.
2
3 This module defines KeyType objects for ML-DSA-65 and ML-KEM-768 without
4 depending on the pqcrypto_fm plugin. These are standalone definitions.
5 """
6
7 from acapy_agent.wallet.key_type import KeyType
8
9
10 # ML-DSA-65 (Dilithium3) - NIST FIPS-204 Digital Signature Algorithm
11 ML_DSA_65 = KeyType(
12     key_type="ml-dsa-65",
13     multicodec_name="ml-dsa-65-pub",
14     multicodec_prefix=b"\xd0\x65",
15     jws_alg="PQC-DSA",
16 )
17
18 # ML-KEM-768 (Kyber768) - NIST FIPS-203 Key Encapsulation Mechanism
19 ML_KEM_768 = KeyType(
20     key_type="ml-kem-768",
21     multicodec_name="ml-kem-768-pub",
22     multicodec_prefix=b"\xe0\x18",
23     jws_alg=None, # KEM algorithms don't have JWS algorithms
24 )

```

Anhang 4.1.14: key_type_patches.py

Listing A-26

pqc_didpeer4_fm - key_type_patches.py

```

1 """Patches for KeyTypes Registry and API Schema Validation.
2
3 This module extends ACA-Py's KeyTypes registry and API schemas to support
4 PQC key types (ML-DSA-65, ML-KEM-768).
5 """
6
7 import logging
8
9 LOGGER = logging.getLogger(__name__)
10
11
12 def register_pqc_key_types(context):
13     """Register PQC KeyTypes in ACA-Py's global KeyTypes registry.
14
15     Args:
16         context: InjectionContext to inject KeyTypes singleton
17
18     This allows all ACA-Py components to look up PQC key types via:
19     - key_types.from_key_type("ml-dsa-65")
20     - key_types.from_multicodec_name("ml-dsa-65-pub")
21     - key_types.from_multicodec_prefix(b"\xd0\x65")
22 """
23     from acapy_agent.wallet.key_type import KeyTypes
24
25     from .key_types import ML_DSA_65, ML_KEM_768
26
27     try:
28         key_types = context.inject(KeyTypes)
29
30         # Register PQC key types
31         key_types.register(ML_DSA_65)
32         key_types.register(ML_KEM_768)
33
34         LOGGER.info(
35             "Registered PQC key types in KeyTypes registry: ml-dsa-65, ml-kem-768"
36         )
37         print("Registered ML-DSA-65 and ML-KEM-768 in KeyTypes registry")
38
39     except Exception as e:
40         LOGGER.error(f"Failed to register PQC key types: {e}")
41         raise
42
43
44 def patch_api_key_type_schemas():
45     """Patch Marshmallow Schema __init__ to inject PQC validators at runtime.
46
47     Patches 3 schema classes in acapy_agent/wallet/routes.py:
48     1. DIDSchema.key_type (line 135) - Response schema
49     2. DIDListQueryStringSchema.key_type (line 335) - Query parameter
50     3. DIDCreateOptionsSchema.key_type (line 355) - Request body
51

```

```

52     This allows:
53     - POST /wallet/did/create with {"options": {"key_type": "ml-dsa-65"}}
54     - GET /wallet/did?key_type=ml-dsa-65
55
56     NOTE: We patch __init__() because aiohttp-apispec creates schema instances
57     AFTER plugin setup. Patching _declared_fields doesn't work because
58     Marshmallow copies them during instantiation.
59     """
60
61     from marshmallow import validate
62
63     from acapy_agent.wallet.routes import (
64         DIDCreateOptionsSchema,
65         DIDListQueryStringSchema,
66         DIDSchema,
67     )
68
69     try:
70         # New validator including PQC key types
71         pqc_validator = validate.OneOf(
72             [
73                 "ed25519",
74                 "bls12381g2",
75                 "p256",
76                 "ml-dsa-65",
77                 "ml-kem-768",
78             ]
79         )
80
81         # Patch 1: DIDSchema (response schema for /wallet/did, /wallet/did/create)
82         original_did_init = DIDSchema.__init__
83
83         def patched_did_init(self, *args, **kwargs):
84             original_did_init(self, *args, **kwargs)
85             if "key_type" in self.fields:
86                 self.fields["key_type"].validators = [pqc_validator]
87
88         DIDSchema.__init__ = patched_did_init
89
90         # Patch 2: DIDListQueryStringSchema (query params for GET /wallet/did)
91         original_list_init = DIDListQueryStringSchema.__init__
92
93         def patched_list_init(self, *args, **kwargs):
94             original_list_init(self, *args, **kwargs)
95             if "key_type" in self.fields:
96                 self.fields["key_type"].validators = [pqc_validator]
97
98         DIDListQueryStringSchema.__init__ = patched_list_init
99
100        # Patch 3: DIDCreateOptionsSchema (CRITICAL: request body for POST /wallet/did
101        # /create)
101        original_create_init = DIDCreateOptionsSchema.__init__
102
103        def patched_create_init(self, *args, **kwargs):
104            original_create_init(self, *args, **kwargs)
105            if "key_type" in self.fields:
106                self.fields["key_type"].validators = [pqc_validator]
107

```

```

108     DIDCreateOptionsSchema.__init__ = patched_create_init
109
110     LOGGER.info("Patched API schemas __init__ to accept PQC key_types at runtime")
111     print("Patched API schemas for PQC key_types (ml-dsa-65, ml-kem-768)")
112
113 except Exception as e:
114     LOGGER.error(f"Failed to patch API schemas: {e}")
115     raise
116
117
118 def patch_did_peer4_supported_key_types(context):
119     """Extend PEER4 DIDMethod to support PQC key types.
120
121     By default, PEER4 only supports [ED25519, X25519].
122     This patch adds [ML_DSA_65, ML_KEM_768] to make did:peer:4 PQC-capable.
123
124     This is required because wallet_create_did() checks:
125         if not method.supports_key_type(key_type):
126             raise HTTPForbidden(...)
127
128     Args:
129         context: InjectionContext (unused, for API consistency)
130     """
131     from acapy_agent.wallet.did_method import PEER4
132
133     from .key_types import ML_DSA_65, ML_KEM_768
134
135     try:
136         # Extend PEER4's supported key types
137         # Original: [ED25519, X25519]
138         # After patch: [ED25519, X25519, ML_DSA_65, ML_KEM_768]
139         PEER4._supported_key_types.extend([ML_DSA_65, ML_KEM_768])
140
141         supported_types = [kt.key_type for kt in PEER4.supported_key_types]
142         LOGGER.info(f"Extended PEER4 to support PQC key types: {supported_types}")
143         print(f"PEER4 now supports: {supported_types}")
144
145     except Exception as e:
146         LOGGER.error(f"Failed to patch PEER4 supported key types: {e}")
147         raise
148
149
150 def patch_alg_mappings_for_pqc():
151     """Extend ALG_MAPPINGS in wallet/keys/manager.py for PQC multikey support.
152
153     ALG_MAPPINGS is used by verkey_to_multiclient() and multiclient_to_verkey()
154     to convert between base58 verkeys and multibase multiclients.
155
156     Without this patch, wallet.create_key(ML_DSA_65) fails with:
157         KeyError: 'ml-dsa-65' in ALG_MAPPINGS[alg]["prefix_hex"]
158     """
159     from acapy_agent.wallet.keys.manager import ALG_MAPPINGS
160
161     from .key_types import ML_DSA_65, ML_KEM_768
162
163     try:
164         # Add PQC algorithms to ALG_MAPPINGS

```

```

165     # Based on multicodec prefixes from pqc_multicodec.py
166     ALG_MAPPINGS["ml-dsa-65"] = {
167         "key_type": ML_DSA_65,
168         "multikey_prefix": "z6MN", # Will be different for ML-DSA, but use z6MN
169         # for now
170         "prefix_hex": "d065", # From PQC_MULTICODECS: b"\xd0\x65"
171         "prefix_length": 2,
172     }
173
174     ALG_MAPPINGS["ml-kem-768"] = {
175         "key_type": ML_KEM_768,
176         "multikey_prefix": "z6Ls", # Will be different for ML-KEM
177         "prefix_hex": "e018", # From PQC_MULTICODECS: b"\xe0\x18"
178         "prefix_length": 2,
179     }
180
181     LOGGER.info("Extended ALG_MAPPINGS for PQC multikey conversion")
182     print("ALG_MAPPINGS extended for PQC (ml-dsa-65, ml-kem-768)")
183
184 except Exception as e:
185     LOGGER.error(f"Failed to patch ALG_MAPPINGS: {e}")
186     raise

```

Anhang 4.1.15: `multicodec_patch.py`

Listing A-27

pqc_didpeer4_fm - multicodec_patch.py

```

1 """Patch ACA-Py's SupportedCodecs to recognize PQC multicodecs.
2
3 This module monkey-patches acapy_agent.utils.multiformats.multicodec.SupportedCodecs
4 to support ML-DSA-65 and ML-KEM-768 multicodec prefixes used in PQC multikeys.
5
6 Without this patch, ACA-Py's multicodec.unwrap() throws:
7     ValueError: Unsupported multicodec
8 when encountering PQC keys in did:peer:4 documents.
9 """
10
11 import logging
12 from typing import Optional
13
14 LOGGER = logging.getLogger(__name__)
15
16
17 def patch_supported_codecs():
18     """Monkey-patch SupportedCodecs to support PQC multicodecs.
19
20     This patches two methods:
21     1. SupportedCodecs.for_data() - Used when unwrapping multicodec-prefixed keys
22     2. SupportedCodecs.by_name() - Used when looking up codecs by name
23
24     Both methods fall back to PQC_MULTICODECS registry if classical codecs
25     don't match.

```

```

26     """
27     try:
28         from acapy_agent.utils.multiformats.multicodec import (
29             SupportedCodecs,
30             Multicodec,
31         )
32         from .pqc_multicodec import PQC_MULTICODECS
33
34     # Save original methods
35     original_for_data = SupportedCodecs.for_data
36     original_by_name = SupportedCodecs.by_name
37
38     @classmethod
39     def for_data_pqc(cls, data: bytes) -> Multicodec:
40         """Enhanced for_data() that supports PQC multicodecs.
41
42         Args:
43             data: Multicodec-prefixed key bytes
44
45         Returns:
46             Multicodec object
47
48         Raises:
49             ValueError: If multicodec prefix is unknown
50         """
51
52     # Try classical codecs first (ED25519, X25519, etc.)
53     try:
54         return original_for_data(data)
55     except ValueError:
56         # Classical codec not found, try PQC registry
57         for codec_name, prefix in PQC_MULTICODECS.items():
58             if data.startswith(prefix):
59                 LOGGER.debug(f"Matched PQC multicodec: {codec_name}")
60                 return Multicodec(codec_name, prefix)
61
62         # Neither classical nor PQC codec matched
63         prefix_hex = data[:2].hex() if len(data) >= 2 else "empty"
64         raise ValueError(
65             f"Unsupported multicodec (prefix: 0x{prefix_hex}). "
66             f"Supported PQC: {list(PQC_MULTICODECS.keys())}"
67         )
68
69     @classmethod
70     def by_name_pqc(cls, name: str) -> Multicodec:
71         """Enhanced by_name() that supports PQC multicodecs.
72
73         Args:
74             name: Multicodec name (e.g., "ml-dsa-65-pub")
75
76         Returns:
77             Multicodec object
78
79         Raises:
80             ValueError: If multicodec name is unknown
81
82     # Try classical codecs first
83     try:

```

```

83         return original_by_name(name)
84     except ValueError:
85         # Classical codec not found, try PQC registry
86         if name in PQC_MULTICODECS:
87             LOGGER.debug(f"Matched PQC multicodec by name: {name}")
88             return Multicodec(name, PQC_MULTICODECS[name])
89
90         # Neither classical nor PQC codec matched
91         raise ValueError(
92             f"Unsupported multicodec: {name}. "
93             f"Supported PQC: {list(PQC_MULTICODECS.keys())}"))
94
95
96     # Apply monkey patches
97     SupportedCodecs.for_data = for_data_pqc
98     SupportedCodecs.by_name = by_name_pqc
99
100    LOGGER.info("Patched SupportedCodecs.for_data() for PQC support")
101    LOGGER.info("Patched SupportedCodecs.by_name() for PQC support")
102
103    except ImportError as e:
104        LOGGER.error(f"Failed to import multicodec module: {e}")
105        raise
106    except Exception as e:
107        LOGGER.error(f"Unexpected error while patching SupportedCodecs: {e}")
108        raise
109
110
111 def unpatch_supported_codecs():
112     """Remove monkey patches and restore original SupportedCodecs methods.
113
114     This function is provided for testing purposes. In normal operation,
115     the patches should remain active.
116     """
117
118     # Note: Restoration would require storing original methods globally
119     # For now, this is a placeholder for API consistency
120     LOGGER.warning("unpatch_supported_codecs() not implemented - patches are permanent
121     ")

```

Anhang 4.1.16: setup.py

Listing A-28

pqc_didpeer4_fm - setup.py

```

1 """Setup for pqc_didpeer4_fm plugin."""
2
3 from setuptools import setup, find_packages
4
5 with open("README.md", "r", encoding="utf-8") as fh:
6     long_description = fh.read()
7
8 setup(
9     name="pqc_didpeer4_fm",

```

```

10     version="0.1.0",
11     author="Ferris Menzel",
12     author_email="admin@example.com",
13     description="Post-Quantum did:peer:4 plugin with ML-DSA-65 + ML-KEM-768 for ACA-Py
14         ",
15     long_description=long_description,
16     long_description_content_type="text/markdown",
17     url="",
18     packages=find_packages(),
19     install_requires=[
20         # Note: aries-cloudagent is NOT listed here because this plugin
21         # runs inside an existing ACA-Py installation which provides all
22         # core dependencies. Only list additional dependencies needed by
23         # this plugin that are not part of standard ACA-Py.
24         "did-peer-4>=0.1.4",
25         "pydid>=0.4.0",
26         "multiformats>=0.3.0",
27         "base58>=2.1.0",
28         "liboqs-python>=0.10.0",
29     ],
30     entry_points={
31         "aries_cloudagent.plugins": [
32             "pqc_didpeer4_fm = pqc_didpeer4_fm:setup"
33         ]
34     },
35     python_requires ">=3.9",
36     classifiers=[
37         "Development Status :: 3 - Alpha",
38         "Intended Audience :: Developers",
39         "Topic :: Security :: Cryptography",
40         "License :: OSI Approved :: Apache Software License",
41         "Programming Language :: Python :: 3",
42         "Programming Language :: Python :: 3.9",
43         "Programming Language :: Python :: 3.10",
44         "Programming Language :: Python :: 3.11",
45         "Programming Language :: Python :: 3.12",
46         "Operating System :: OS Independent",
47     ],
48     keywords="pqc post-quantum cryptography did peer ml-dsa ml-kem acapy aries ssi",
49 )

```

Anhang 4.1.17: README.md

Listing A-29

pqc_didpeer4_fm - README.md

```

1 # pqc_didpeer4_fm
2
3 **Post-Quantum did:peer:4 Plugin for ACA-Py**
4
5 Transparently replaces ED25519/X25519 with **ML-DSA-65** (NIST FIPS-204) and **ML-KEM
6 -768** (NIST FIPS-203) in did:peer:4 DIDs.

```

```

7  ## Features
8
9 - **Post-Quantum Security**: ML-DSA-65 for signatures, ML-KEM-768 for key agreement
10 - **Transparent Integration**: NO API changes needed - works with existing workflows
11 - **Zero Code Changes**: Existing notebooks/scripts continue to work unchanged
12 - **Standards Compliant**: NIST FIPS-203/204, W3C Multicodec (provisional)
13 - **DIDComm v2 Ready**: Full encryption and authentication support
14
15 ## Installation
16
17 ````bash
18 cd acapy-plugins/pqc_didpeer4_fm
19 pip install -e .
20 `````
21
22 ## Usage
23
24 ### 1. Load Plugin in ACA-Py
25
26 ````bash
27 aca-py start \
28   --plugin pqc_didpeer4_fm \
29   --endpoint https://agent.example.com:8020 \
30   --admin 0.0.0.0 8021 \
31 ...
32 `````
33
34 ### 2. docker-compose.yml
35
36 ````yaml
37 services:
38   issuer:
39     command: >
40       start
41       --plugin pqc_didpeer4_fm # <-- Add this line
42       --endpoint https://host.docker.internal:8020
43     ...
44 `````
45
46 ### 3. Create Connection (Unchanged!)
47
48 ````python
49 # Existing code continues to work!
50 invitation_data = {
51   "use_did_method": "did:peer:4", # <-- Plugin makes this PQC automatically
52   "handshake_protocols": ["https://didcomm.org/didexchange/1.1"],
53   "my_label": "My Agent"
54 }
55
56 response = requests.post(
57   "http://localhost:8021/out-of-band/create-invitation",
58   json=invitation_data
59 )
60
61 # Plugin creates did:peer:4 with ML-DSA-65 + ML-KEM-768 automatically!
62 `````
63

```

```

64 | ##### 4. View DIDs
65 |
66 | ````python
67 | # GET /wallet/did now shows PQC keys
68 | response = requests.get("http://localhost:8021/wallet/did")
69 |
70 | # Response:
71 | # {
72 | #   "results": [
73 | #     "did": "did:peer:4:z6MNxxx...",
74 | #     "key_type": "ml-dsa-65", # <-- Shows PQC!
75 | #     "method": "did:peer:4",
76 | #     "metadata": {
77 | #       "pqc_enabled": true,
78 | #       "signature_algorithm": "ml-dsa-65",
79 | #       "key_agreement_algorithm": "ml-kem-768",
80 | #       "plugin": "pqc_didpeer4_fm"
81 | #     }
82 | #   ]
83 | # }
84 | ````

85 |
86 | ## How It Works
87 |
88 | #### Transparent Monkey-Patching
89 |
90 | The plugin patches `BaseConnectionManager.create_did_peer_4()` at runtime:
91 |
92 | ````

93 | Workflow: POST /out-of-band/create-invitation {"use_did_method": "did:peer:4"}
94 |           |
95 |           V
96 |           Out-of-Band Manager
97 |           |
98 |           V
99 |           BaseConnectionManager.create_did_peer_4()
100 |           |
101 |           V
102 |           PLUGIN INTERCEPTS HERE
103 |           |
104 |           V
105 |           Creates ML-DSA-65 + ML-KEM-768 keys instead of ED25519 + X25519
106 |           |
107 |           V
108 |           did:peer:4:z6MNxxx... (PQC enabled)
109 | ````

110 |
111 | #### Two Keys, Two Purposes
112 |
113 | Key Type | Purpose | DID Document Relationship |
114 | ----- | ----- | ----- |
115 | **ML-DSA-65** | Digital Signatures | `authentication`, `assertionMethod` |
116 | **ML-KEM-768** | Key Agreement (Encryption) | `keyAgreement` |
117 |
118 | #### Why Both?
119 |
120 | - **ML-DSA-65** proves authenticity ("I am really Alice")

```

```

121 - **ML-KEM-768** enables encryption ("Only Bob can read this")
122 - **Both are required** for secure DIDComm messaging
123
124 ## Technical Details
125
126 ### Multicodec Prefixes (Provisional)
127
128 - ML-DSA-65: '0xd065' --> Multikey prefix 'z6MN'
129 - ML-KEM-768: '0xe018' --> Multikey prefix 'z6MK768'
130
131 ### Dependencies
132
133 - 'aries-cloudagent~=1.0.0'
134 - 'did-peer-4>=0.1.4'
135 - 'pydid>=0.4.0'
136 - 'multiformats>=0.3.0'
137 - 'base58>=2.1.0'
138 - 'pqcrypto_fm' (PQC crypto plugin - must be installed separately)
139
140 ### Workflow Integration
141
142 **NO CODE CHANGES NEEDED!**
143
144 Existing workflows (e.g., 'SSI_Complete_Workflow.ipynb' Cells 13-16) continue to work:
145
146 ````python
147 # Cell 13 - Connection Creation (unchanged)
148 invitation_response = api_post(
149     ISSUER_ADMIN_URL,
150     "/out-of-band/create-invitation",
151     {"use_did_method": "did:peer:4", ...}
152 )
153 # Plugin automatically uses ML-DSA-65 + ML-KEM-768
154
155 # Cell 16 - DID Overview (unchanged)
156 dids = api_get(ISSUER_ADMIN_URL, "/wallet/did")
157 # Shows "key_type": "ml-dsa-65" automatically
158 ````

159
160 ## Important Notes
161
162 1. **All agents must have the plugin**: If one agent has PQC enabled, all connected
   agents must also have it
163 2. **Not backward compatible**: Classic did:peer:4 (ED25519) and PQC did:peer:4 (ML-
   DSA-65) cannot interoperate
164 3. **Experimental**: PQC multicodec prefixes are provisional (W3C draft)
165 4. **Requires OpenSSL 3.5+**: For native ML-DSA/ML-KEM support
166
167 ## References
168
169 - [NIST FIPS-203 (ML-KEM)](https://csrc.nist.gov/pubs/fips/203/final)
170 - [NIST FIPS-204 (ML-DSA)](https://csrc.nist.gov/pubs/fips/204/final)
171 - [did:peer:4 Specification] (https://identity.foundation/peer-did-method-spec/)
172 - [W3C Multicodec Registry] (https://w3c-ccg.github.io/multicodec/)
173
174 ## License
175

```

```

176 Apache License 2.0
177
178 ## Author
179
180 Ferris Menzel

```

Anhang 4.2: Dockerfile: acapy-base-pqc

Listing A-30

Dockerfile - acapy-base-pqc

```

1 # =====
2 # Stage 1: Build OpenSSL 3.5.4 with native PQC support (ML-KEM, ML-DSA)
3 # =====
4 FROM debian:bookworm-slim AS openssl-builder
5
6 ARG OPENSSL_VERSION=3.5.4
7 ARG OPENSSL_PREFIX=/usr/local/ssl
8
9 # Install build dependencies
10 RUN apt-get update && \
11     apt-get install -y --no-install-recommends \
12     build-essential \
13     ca-certificates \
14     curl \
15     libssl-dev \
16     perl \
17     && rm -rf /var/lib/apt/lists/*
18
19 # Download and extract OpenSSL 3.5.4
20 WORKDIR /tmp
21 RUN curl -fsSL "https://www.openssl.org/source/openssl-${OPENSSL_VERSION}.tar.gz" -o \
22     openssl.tar.gz && \
23     tar -xzf openssl.tar.gz && \
24     rm openssl.tar.gz
25
26 # Build OpenSSL with native PQC support
27 WORKDIR /tmp/openssl-${OPENSSL_VERSION}
28 RUN ./Configure \
29     --prefix=${OPENSSL_PREFIX} \
30     --openssldir=${OPENSSL_PREFIX}/ssl \
31     shared \
32     enable-fips \
33     linux-x86_64 && \
34     make -j"${nproc}" && \
35     make install_sw install_ssldirs
36
37 # Verify OpenSSL 3.5.4 was built successfully
38 # RUN LD_LIBRARY_PATH=${OPENSSL_PREFIX}/lib64 ${OPENSSL_PREFIX}/bin/openssl version &&
39 #     ${OPENSSL_PREFIX}/bin/openssl list -kem-algorithms | grep -i ml-kem && \
40 #     ${OPENSSL_PREFIX}/bin/openssl list -signature-algorithms | grep -i ml-dsa

```

```

41 RUN LD_LIBRARY_PATH=${OPENSSL_PREFIX}/lib64 ${OPENSSL_PREFIX}/bin/openssl version
42
43 # =====
44 # Stage 2: Build liboqs for PQC support
45 # =====
46 FROM debian:bookworm-slim AS liboqs-builder
47
48 ARG LIBOQS_VERSION=0.14.0
49 ARG LIBOQS_PREFIX=/usr/local
50
51 # Install build dependencies
52 RUN apt-get update && \
53     apt-get install -y --no-install-recommends \
54     build-essential \
55     ca-certificates \
56     cmake \
57     curl \
58     git \
59     libssl-dev \
60     ninja-build \
61     && rm -rf /var/lib/apt/lists/*
62
63 # Download and build liboqs
64 WORKDIR /tmp
65 RUN curl -fsSL "https://github.com/open-quantum-safe/liboqs/archive/refs/tags/${LIBOQS_VERSION}.tar.gz" -o liboqs.tar.gz && \
66     tar -xzf liboqs.tar.gz && \
67     rm liboqs.tar.gz
68
69 WORKDIR /tmp/liboqs-${LIBOQS_VERSION}
70 RUN mkdir build && cd build && \
71     cmake -GNinja \
72     -DCMAKE_INSTALL_PREFIX=${LIBOQS_PREFIX} \
73     -DBUILD_SHARED_LIBS=ON \
74     .. && \
75     ninja && \
76     ninja install
77
78 # Verify liboqs installation
79 RUN ldconfig && \
80     ls -la ${LIBOQS_PREFIX}/lib/liboqs.* && \
81     echo "liboqs ${LIBOQS_VERSION} built successfully"
82
83 # =====
84 # Stage 3: ACA-Py Build Stage (Poetry)
85 # =====
86 FROM python:3.12-slim-bookworm AS build
87 ARG python_version=3.12
88
89 RUN pip install --no-cache-dir poetry==2.1.1
90
91 WORKDIR /src
92
93 COPY ./pyproject.toml ./poetry.lock ./
94 RUN poetry install --no-root
95
96 COPY ./acapy_agent ./acapy_agent

```

```

97 COPY ./README.md /src
98 RUN poetry build
99
100 # =====
101 # Stage 4: Main Runtime Image with OpenSSL 3.5.4 PQC + liboqs
102 # =====
103 FROM python:3.12-slim-bookworm AS main
104
105 ARG python_version=3.12
106 ARG uid=1001
107 ARG user=aries
108 ARG acapy_name="acapy-agent"
109 ARG acapy_version
110 ARG acapy_reqs=[didcommv2]
111 ARG OPENSSL_PREFIX=/usr/local/ssl
112
113 ENV HOME="/home/$user" \
114     APP_ROOT="/home/$user" \
115     LC_ALL=C.UTF-8 \
116     LANG=C.UTF-8 \
117     PIP_NO_CACHE_DIR=off \
118     PYTHONUNBUFFERED=1 \
119     PYTHONIOENCODING=UTF-8 \
120     RUST_LOG=warn \
121     SHELL=/bin/bash \
122     SUMMARY="$acapy_name image with PQC support" \
123     DESCRIPTION="$acapy_name provides a base image for running acapy agents with Post- \
124         Quantum Cryptography support via OpenSSL 3.5.4. \
125     Supports ML-KEM-768, ML-DSA-65, and X25519MLKEM768 hybrid algorithms. Based on \
126         Debian Bookworm."
127
128 LABEL summary="$SUMMARY" \
129     description="$DESCRIPTION" \
130     io.k8s.description="$DESCRIPTION" \
131     io.k8s.display-name="$acapy_name $acapy_version" \
132     name=$acapy_name \
133     acapy.version="$acapy_version" \
134     maintainer=""
135
136 # Copy OpenSSL 3.5.4 from builder stage
137 COPY --from=openssl-builder ${OPENSSL_PREFIX} ${OPENSSL_PREFIX}
138
139 # Copy liboqs from builder stage
140 COPY --from=liboqs-builder /usr/local/lib/liboqs.* /usr/local/lib/
141 COPY --from=liboqs-builder /usr/local/include/oqs /usr/local/include/oqs
142
143 # Copy PQC Root CA certificate
144 COPY hopE/pqc_sidecarproxy_nginx/certs/rootCA.crt /usr/local/share/ca-certificates/pqc-
145             -root-ca.crt
146
147 # Add aries user
148 RUN useradd -U -ms /bin/bash -u $uid $user
149
150 # Install environment (without openssl, we use custom build)
151 RUN apt-get update && \
152     apt-get install -y --no-install-recommends \
153     apt-transport-https \

```

```

151      ca-certificates \
152      curl \
153      git \
154      libffi-dev \
155      libgmp10 \
156      libncurses5 \
157      libncursesw5 \
158      sqlite3 \
159      zlib1g && \
160      apt-get autoremove -y && \
161      apt-get clean -y && \
162      rm -rf /var/lib/apt/lists/* /usr/share/doc/*
163
164 # Import PQC Root CA into system trust store
165 RUN update-ca-certificates && \
166     echo "PQC Root CA imported successfully"
167
168 # Replace system OpenSSL with PQC-enabled version and configure liboqs
169 RUN ln -sf ${OPENSSL_PREFIX}/bin/openssl /usr/bin/openssl && \
170     ln -sf ${OPENSSL_PREFIX}/lib64/libssl.so.3 /usr/lib/x86_64-linux-gnu/libssl.so.3
171     && \
172     ln -sf ${OPENSSL_PREFIX}/lib64/libcrypto.so.3 /usr/lib/x86_64-linux-gnu/libcrypto.
173     so.3 && \
174     echo "${OPENSSL_PREFIX}/lib64" > /etc/ld.so.conf.d/openssl-pqc.conf && \
175     echo "/usr/local/lib" > /etc/ld.so.conf.d/liboqs.conf && \
176     ldconfig
177
178 # Link OpenSSL's cert directory to system certs (for Python ssl module)
179 # This ensures Python's ssl module finds the PQC Root CA
180 RUN rm -rf ${OPENSSL_PREFIX}/ssl/certs && \
181     ln -s /etc/ssl/certs ${OPENSSL_PREFIX}/ssl/certs && \
182     ln -s /etc/ssl/certs/ca-certificates.crt ${OPENSSL_PREFIX}/ssl/cert.pem
183
184 # Set SSL environment variables for Python ssl module
185 ENV SSL_CERT_FILE=/etc/ssl/certs/ca-certificates.crt \
186     SSL_CERT_DIR=/etc/ssl/certs \
187     REQUESTS_CA_BUNDLE=/etc/ssl/certs/ca-certificates.crt \
188     CURL_CA_BUNDLE=/etc/ssl/certs/ca-certificates.crt
189
190 # Verify OpenSSL installation and PQC support
191 RUN openssl version && \
192     openssl list -kem-algorithms | grep -i ml-kem && \
193     echo "OpenSSL 3.5.4 with PQC support successfully installed"
194
195 WORKDIR $HOME
196
197 # Add local binaries and aliases to path
198 ENV PATH="$HOME/.local/bin:$PATH"
199
200 # - In order to drop the root user, we have to make some directories writable
201 #   to the root group as OpenShift default security model is to run the container
202 #   under random UID.
203 RUN usermod -a -G 0 $user
204
205 # Create standard directories to allow volume mounting and set permissions
206 # Note: PIP_NO_CACHE_DIR environment variable should be cleared to allow caching
207 RUN mkdir -p \

```

```

206     $HOME/.acapy_agent \
207     $HOME/.cache/pip/http \
208     $HOME/.indy_client \
209     $HOME/ledger/sandbox/data \
210     $HOME/log
211
212 # The root group needs access the directories under $HOME/.indy_client and $HOME/ .
213 # acapy_agent for the container to function in OpenShift.
214 RUN chown -R $user:root $HOME/.indy_client $HOME/.acapy_agent && \
215     chmod -R ug+rw $HOME/log $HOME/ledger $HOME/.acapy_agent $HOME/.cache $HOME/ .
216     indy_client
217
218 # Create /home/indy and symlink .indy_client folder for backwards compatibility with
219 # artifacts created on older indy-based images.
220 RUN mkdir -p /home/indy
221 RUN ln -s /home/aries/.indy_client /home/indy/.indy_client
222
223 # Install ACA-py from the wheel as $user,
224 # and ensure the permissions on the python 'site-packages' and $HOME/.local folders
225 # are set correctly.
226 USER $user
227 COPY --from=build /src/dist/acapy_agent*.whl .
228 RUN acapy_agent_package=$(find ./ -name "acapy_agent*.whl" | head -n 1) && \
229     echo "Installing ${acapy_agent_package} ..." && \
230     pip install --no-cache-dir --find-links=. ${acapy_agent_package}${acapy_reqs} && \
231     rm acapy_agent*.whl && \
232     chmod +rx $(python -m site --user-site) $HOME/.local
233
234 # Install pqc_didpeer4_fm plugin
235 COPY --chown=$user:root acapy-plugins/pqc_didpeer4_fm /tmp/pqc_didpeer4_fm
236 RUN pip install --no-cache-dir /tmp/pqc_didpeer4_fm && \
237     rm -rf /tmp/pqc_didpeer4_fm && \
238     echo "pqc_didpeer4_fm plugin installed successfully"
239
240 ENTRYPOINT ["aca-py"]

```

Anhang 4.3: docker-compose.yml: SSI Agenten mit acapy-base-pqc & Plugin

Listing A-31

multicodec_patch.py

```

1 version: '3.8'
2
3 services:
4     # unter command: > --label XY ==> --plugin pqc_didpeer4_fm ergänzen für PQC
5     issuer:
6         build:
7             context: ..
8             dockerfile: hopE/Dockerfile.acapy-base-pqc
9             image: acapy-base-pqc
10            container_name: issuer-agent

```

```

11      environment:
12        - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
13        - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
14        - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
15        - PUBLIC_TAILS_URL=https://host.docker.internal:6543
16        - TAILS_FILE_COUNT=100
17    networks:
18      - hope-issuer
19    volumes:
20      - issuer-data:/home/aries/.acapy_agent
21    extra_hosts:
22      - "host.docker.internal:host-gateway"
23    command: >
24      start
25      --label "Issuer Agent"
26      --plugin pqc_didpeer4_fm
27      --inbound-transport http 0.0.0.0 8020
28      --outbound-transport http
29      --endpoint https://host.docker.internal:8020
30      --admin 0.0.0.0 8021
31      --admin-insecure-mode
32      --auto-provision
33      --wallet-type askar
34      --wallet-name issuer_wallet
35      --wallet-key issuer_wallet_key_000000000000
36      --genesis-url https://host.docker.internal:8000/genesis
37      --log-level info
38      --auto-accept-invites
39      --auto-accept-requests
40      --auto-ping-connection
41      --auto-respond-credential-proposal
42      --auto-respond-credential-offer
43      --auto-respond-credential-request
44      --auto-verify-presentation
45      --public-invites
46      --preserve-exchange-records
47      --tails-server-base-url https://host.docker.internal:6543
48      --notify-revocation
49    healthcheck:
50      test: ["CMD", "curl", "-f", "http://localhost:8021/status/ready"]
51      interval: 30s
52      timeout: 10s
53      retries: 5
54      start_period: 30s
55
56  # unter command: > --label XY ==> --plugin pqc_didpeer4_fm ergänzen für PQC
57  holder:
58    build:
59      context: ..
60      dockerfile: hopE/Dockerfile.acapy-base-pqc
61    image: acapy-base-pqc
62    container_name: holder-agent
63    environment:
64      - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
65      - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
66      - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
67      - PUBLIC_TAILS_URL=https://host.docker.internal:6543

```

```

68     networks:
69         - hope-holder
70     volumes:
71         - holder-data:/home/aries/.acapy_agent
72     extra_hosts:
73         - "host.docker.internal:host-gateway"
74     command: >
75         start
76         --label "Holder Agent"
77         --plugin pqc_didpeer4_fm
78         --inbound-transport http 0.0.0.0 8030
79         --outbound-transport http
80         --endpoint https://host.docker.internal:8030
81         --admin 0.0.0.0 8031
82         --admin-insecure-mode
83         --auto-provision
84         --wallet-type askar
85         --wallet-name holder_wallet
86         --wallet-key holder_wallet_key_000000000000
87         --genesis-url https://host.docker.internal:8000/genesis
88         --log-level info
89         --auto-accept-invites
90         --auto-accept-requests
91         --auto-ping-connection
92         --auto-respond-credential-offer
93         --auto-store-credential
94         --public-invites
95         --tails-server-base-url https://host.docker.internal:6543
96         --preserve-exchange-records
97     healthcheck:
98         test: ["CMD", "curl", "-f", "http://localhost:8031/status/ready"]
99         interval: 30s
100        timeout: 10s
101        retries: 5
102        start_period: 30s
103
104 # unter command: > --label XY ==> --plugin pqc_didpeer4_fm ergänzen für PQC
105 verifier:
106     build:
107         context: ..
108         dockerfile: hopE/Dockerfile.acapy-base-pqc
109     image: acapy-base-pqc
110     container_name: verifier-agent
111     environment:
112         - DOCKERHOST=${DOCKERHOST:-host.docker.internal}
113         - GENESIS_URL=${GENESIS_URL:-https://host.docker.internal:8000/genesis}
114         - LEDGER_URL=${LEDGER_URL:-https://host.docker.internal:8000}
115         - PUBLIC_TAILS_URL=https://host.docker.internal:6543
116     networks:
117         - hope-verifier
118     volumes:
119         - verifier-data:/home/aries/.acapy_agent
120     extra_hosts:
121         - "host.docker.internal:host-gateway"
122     command: >
123         start
124         --label "Verifier Agent"

```

```

125      --plugin pqc_didpeer4_fm
126      --inbound-transport http 0.0.0.0 8040
127      --outbound-transport http
128      --endpoint https://host.docker.internal:8040
129      --admin 0.0.0.0 8041
130      --admin-insecure-mode
131      --auto-provision
132      --wallet-type askar
133      --wallet-name verifier_wallet
134      --wallet-key verifier_wallet_key_000000000000
135      --genesis-url https://host.docker.internal:8000/genesis
136      --log-level info
137      --auto-accept-invites
138      --auto-accept-requests
139      --auto-ping-connection
140      --auto-verify-presentation
141      --public-invites
142      --tails-server-base-url https://host.docker.internal:6543
143      --preserve-exchange-records
144
healthcheck:
145     test: ["CMD", "curl", "-f", "http://localhost:8041/status/ready"]
146     interval: 30s
147     timeout: 10s
148     retries: 5
149     start_period: 30s
150
151 # Post-Quantum Nginx Reverse Proxy für Issuer Agent
152 pqc-sidecarproxy-issuer:
153     build:
154         context: ./pqc_sidecarproxy_nginx
155         dockerfile: Dockerfile
156     args:
157         OPENSSL_TAG: openssl-3.5.4
158         LIBOQS_TAG: 0.13.0
159         OQSPROVIDER_TAG: 0.9.0
160         NGINX_VERSION: 1.28.0
161         SIG_ALG: mldsa65
162         DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
163     container_name: pqc-sidecarproxy-issuer
164     environment:
165         # OpenSSL Configuration
166         - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
167         # Post-Quantum Key Exchange Groups
168         - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
169     networks:
170         - von_sidecarproxy
171         - hope-issuer
172     ports:
173         - "8020:8020" # Issuer Inbound Transport HTTPS (ML-KEM-768)
174         - "8021:8021" # Issuer Admin API HTTPS (ML-KEM-768)
175     volumes:
176         # Custom nginx configuration for reverse proxy
177         - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_issuer.conf:/opt/nginx/nginx-conf/
178             nginx.conf:ro
179         # Custom ML-DSA-65 certificates
180         - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
181         # Logs

```

```

181      - nginx-logs:/opt/nginx/logs
182 depends_on:
183   - issuer
184   - holder
185   - verifier
186 restart: unless-stopped
187 healthcheck:
188   test: ["CMD", "curl", "-k", "-f", "https://localhost:8021/health"]
189   interval: 30s
190   timeout: 10s
191   retries: 5
192   start_period: 10s
193
194 # Post-Quantum Nginx Reverse Proxy für Holder Agent
195 pqc-sidecarproxy-holder:
196   build:
197     context: ./pqc_sidecarproxy_nginx
198     dockerfile: Dockerfile
199     args:
200       OPENSSL_TAG: openssl-3.5.4
201       LIBOQS_TAG: 0.13.0
202       OQSPROVIDER_TAG: 0.9.0
203       NGINX_VERSION: 1.28.0
204       SIG_ALG: mldsa65
205       DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
206   container_name: pqc-sidecarproxy-holder
207   environment:
208     # OpenSSL Configuration
209     - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
210     # Post-Quantum Key Exchange Groups
211     - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
212   networks:
213     - von_sidecarproxy
214     - hope-holder
215   ports:
216     - "8030:8030" # Holder Inbound Transport HTTPS (ML-KEM-768)
217     - "8031:8031" # Holder Admin API HTTPS (ML-KEM-768)
218   volumes:
219     # Custom nginx configuration for reverse proxy
220     - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_holder.conf:/opt/nginx/nginx-conf/
221           nginx.conf:ro
222     # Custom ML-DSA-65 certificates
223     - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
224     # Logs
225     - nginx-logs:/opt/nginx/logs
226   depends_on:
227     - issuer
228     - holder
229     - verifier
230   restart: unless-stopped
231   healthcheck:
232     test: ["CMD", "curl", "-k", "-f", "https://localhost:8031/health"]
233     interval: 30s
234     timeout: 10s
235     retries: 5
236     start_period: 10s

```

```

237 # Post-Quantum Nginx Reverse Proxy für Verifier Agent
238 pqc-sidecarproxy-verifier:
239   build:
240     context: ./pqc_sidecarproxy_nginx
241     dockerfile: Dockerfile
242   args:
243     OPENSSL_TAG: openssl-3.5.4
244     LIBOQS_TAG: 0.13.0
245     OQSPROVIDER_TAG: 0.9.0
246     NGINX_VERSION: 1.28.0
247     SIG_ALG: mldsa65
248     DEFAULT_GROUPS: X25519MLKEM768:mlkem768:x25519:mlkem1024
249   container_name: pqc-sidecarproxy-verifier
250   environment:
251     # OpenSSL Configuration
252     - OPENSSL_CONF=/opt/openssl/.openssl/ssl/openssl.cnf
253     # Post-Quantum Key Exchange Groups
254     - DEFAULT_GROUPS=X25519MLKEM768:mlkem768:x25519:mlkem1024
255   networks:
256     - von_sidecarproxy
257     - hope-verifier
258   ports:
259     - "8040:8040" # Verifier Inbound Transport HTTPS (ML-KEM-768)
260     - "8041:8041" # Verifier Admin API HTTPS (ML-KEM-768)
261   volumes:
262     # Custom nginx configuration for reverse proxy
263     - ./pqc_sidecarproxy_nginx/nginx-conf/nginx_verifier.conf:/opt/nginx/nginx-conf/
264       nginx.conf:ro
265     # Custom ML-DSA-65 certificates
266     - ./pqc_sidecarproxy_nginx/certs:/opt/nginx/certs:ro
267     # Logs
268     - nginx-logs:/opt/nginx/logs
269   depends_on:
270     - issuer
271     - holder
272     - verifier
273   restart: unless-stopped
274   healthcheck:
275     test: ["CMD", "curl", "-k", "-f", "https://localhost:8041/health"]
276     interval: 30s
277     timeout: 10s
278     retries: 5
279     start_period: 10s
280   networks:
281     hope-issuer:
282     hope-holder:
283     hope-verifier:
284     von_sidecarproxy:
285       external: true
286
287   volumes:
288     issuer-data:
289     holder-data:
290     verifier-data:
291     nginx-logs:

```

Anhang 4.4: Issuer Agent mit PQC-Plugin Boot Log

Listing A-32

Issuer Agent mit PQC-Plugin Boot Log

```

1 Executing task in folder ferris: docker logs --tail 1000 -f 767
d8558623044fb63e556e0bd1fdce0912ed8be74c26003b449f425d349fa3e
2
3 2025-12-03 23:40:34,654 acapy_agent.config.default_context INFO Registering default
   plugins
4 2025-12-03 23:40:34,772 acapy_agent.config.default_context INFO Registering askar
   plugins
5 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Saved original
   _create_keypair function
6 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Successfully patched
   askar._create_keypair for PQC support
7 =====
8   pqc_didpeer4_fm Plugin v0.1.0
9   Post-Quantum did:peer:4 with ML-DSA-65 + ML-KEM-768
10  by Ferris Menzel
11 =====
12  Askar patched for PQC key generation (liboqs-python)
13 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched Session.
   insert_key() for PQC support
14 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched Session.
   fetch_key() for PQC support
15 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched Session.
   update_key() for PQC support
16 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched AskarWallet.
   assign_kid_to_key() for PQC support
17 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.key_type_patches INFO Registered PQC key
   types in KeyTypes registry: ml-dsa-65, ml-kem-768
18 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.key_type_patches INFO Patched API schemas
   __init__ to accept PQC key_types at runtime
19  Registered ML-DSA-65 and ML-KEM-768 in KeyTypes registry
20  Patched API schemas for PQC key_types (ml-dsa-65, ml-kem-768)
21  PEER4 now supports: ['ed25519', 'x25519', 'ml-dsa-65', 'ml-kem-768']
22 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.key_type_patches INFO Extended PEER4 to
   support PQC key types: ['ed25519', 'x25519', 'ml-dsa-65', 'ml-kem-768']
23  ALG_MAPPINGS extended for PQC (ml-dsa-65, ml-kem-768)
24 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.key_type_patches INFO Extended
   ALG_MAPPINGS for PQC multikey conversion
25  PQC Multicodecs registered (ML-DSA-65, ML-KEM-768)
26 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.multicodec_patch INFO Patched
   SupportedCodecs.for_data() for PQC support
27 2025-12-03 23:40:34,841 pqc_didpeer4_fm.v1_0.multicodec_patch INFO Patched
   SupportedCodecs.by_name() for PQC support
28  SupportedCodecs patched for PQC multicodec decoding
29  DIDComm v1 pack/unpack patched for ML-KEM-768
30  AttachDecorator patched for ML-DSA-65 JWS signatures
31 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched AskarWallet.
   pack_message() for PQC support
32 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched AskarWallet.
   unpack_message() for PQC support
33 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched AskarWallet.
   sign_message() for PQC support

```

```

34| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched AskarWallet.
   verify_message() for PQC support
35| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched
   AttachDecoratorData.sign() for PQC support
36| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.askar_pqc_patch INFO Patched
   AttachDecoratorData.verify() for PQC support
37| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.validator_patch INFO     Patched validator
   in: attach_decorator (module-level), AttachDecoratorDataJWSHeaderSchema.kid field
38| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.validator_patch INFO JWSHeaderKid
   validator patched for did:peer:4 support
39| JWSHeaderKid validator patched for did:peer:4
40| AskarWallet patched for ML-KEM-768 verkey lookup
41| 2025-12-03 23:40:34,842 pqc_didpeer4_fm ERROR WALLET PATCH APPLIED SUCCESSFULLY AT
   PLUGIN LOAD
42| 2025-12-03 23:40:34,842 pqc_didpeer4_fm ERROR     Original: <function AskarWallet.
   get_local_did_for_verkey at 0x73ec177a6f20>
43| 2025-12-03 23:40:34,842 pqc_didpeer4_fm ERROR     Patched: <function
   get_local_did_for_verkey_pqc at 0x73ec172174c0>
44| 2025-12-03 23:40:34,842 pqc_didpeer4_fm.v1_0.connection_target_patch INFO Patching
   ConnectionTargetSchema to accept PQC keys...
45| 2025-12-03 23:40:34,843 pqc_didpeer4_fm.v1_0.connection_target_patch INFO
   recipient_keys field patched to accept PQC keys
46| 2025-12-03 23:40:34,843 pqc_didpeer4_fm.v1_0.connection_target_patch INFO
   routing_keys field patched to accept PQC keys
47| 2025-12-03 23:40:34,843 pqc_didpeer4_fm.v1_0.connection_target_patch INFO     sender_key
   field patched to accept PQC keys
48| 2025-12-03 23:40:34,843 pqc_didpeer4_fm.v1_0.connection_target_patch INFO     New
   pattern: base58 strings with 32+ characters
49| ConnectionTarget schema patched for PQC key validation
50| 2025-12-03 23:40:34,843 pqc_didpeer4_fm.v1_0.connection_target_patch INFO     Accepts
   : ED25519 (43-44 chars), ML-DSA-65 (~2650 chars), ML-KEM-768 (~1600 chars)
51| Monkey patches applied to BaseConnectionManager
   - create_did_peer_4 --> PQC version (eliminates ED25519)
52|   - _extract_key_material_in_base58_format --> PQC support
53|   - long_did_peer_4_to_short --> Preserves PQC key_type
54|   - long_did_peer_to_short --> Handles short-form DIDs correctly
55|   - record_keys_for_resolvable_did --> Stores BOTH PQC keys
56|   - resolve_inbound_connection --> DEBUG logging for diagnostics
57|   - find_connection --> Handles long/short form my_did (credential fix)
58|
59| PQC Peer4 Resolver registered
60| =====
61| pqc_didpeer4_fm loaded successfully!
62| did:peer:4 now uses ML-DSA-65 + ML-KEM-768
63| =====
64|
65| 2025-12-03 23:40:34,923 acapy_agent.config.ledger INFO Fetching genesis transactions
   from: https://host.docker.internal:8000/genesis
66| 2025-12-03 23:40:34,931 acapy_agent.core.profile INFO Create profile manager: askar
67| 2025-12-03 23:40:35,512 acapy_agent.config.wallet INFO Created new profile - Profile
   name: issuer_wallet, backend: askar
68| 2025-12-03 23:40:35,515 acapy_agent.config.wallet INFO No public DID created
69| 2025-12-03 23:40:35,575 acapy_agent.config.ledger INFO Ledger configuration complete
70| 2025-12-03 23:40:35,575 acapy_agent.core.conductor INFO Ledger configured successfully
   .
71| 2025-12-03 23:40:35,584 acapy_agent.core.conductor INFO Wallet type record not found.
72| 2025-12-03 23:40:35,585 acapy_agent.core.conductor INFO New agent. Setting wallet type
   to askar.

```

```

73| 2025-12-03 23:40:35,753 acapy_agent.config.banner INFO
74| ::::::::::::::::::::
75| :: Issuer Agent :::::
76| :: :::::
77| :: :::::
78| :: Inbound Transports:::::
79| :: :::::
80| :: - http://0.0.0.0:8020 :::::
81| :: :::::
82| :: Outbound Transports:::::
83| :: :::::
84| :: - http :::::
85| :: - https :::::
86| :: :::::
87| :: Administration API:::::
88| :: :::::
89| :: - http://0.0.0.0:8021 :::::
90| :: :::::
91| :: ::::: ver: 1.3.2 :::::
92| ::::::::::::::::::::
93|
94| 2025-12-03 23:40:35,753 acapy_agent.config.banner INFO
95| ::::::::::::::::::::
96| :: DEPRECATION NOTICE :::::
97| :: -----
98| :: Receiving a core DIDComm protocol with the 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec' :::
99| :: prefix is deprecated. All parties sending this prefix should be notified that :::
100| :: support for receiving such messages will be removed in a future release. Use :::
101| :: https://didcomm.org/ instead. :::
102| :: -----
103| :: Aries RFC 0160: Connection Protocol is deprecated and support will be removed in :::
104| :: a future release; use RFC 0023: DID Exchange instead. :::
105| :: -----
106| :: Aries RFC 0036: Issue Credential 1.0 is deprecated and support will be removed :::
107| :: in a future release; use RFC 0453: Issue Credential 2.0 instead. :::
108| :: -----
109| :: Aries RFC 0037: Present Proof 1.0 is deprecated and support will be removed in a :::
110| :: future release; use RFC 0454: Present Proof 2.0 instead. :::
111| ::::::::::::::::::::
112|
113| 2025-12-03 23:40:35,754 acapy_agent.core.conductor INFO Wallet version storage record
114| not found.
115| 2025-12-03 23:40:35,754 acapy_agent.core.conductor INFO No upgrade from version was
116| found from wallet or via --from-version startup argument. Defaulting to v0.7.5.
117| 2025-12-03 23:40:35,755 acapy_agent.core.conductor INFO Upgrade configurations
118| available. Initiating upgrade.
119| 2025-12-03 23:40:35,757 acapy_agent.commands.upgrade INFO No ACA-Py version found in
120| wallet storage.
121| 2025-12-03 23:40:35,757 acapy_agent.commands.upgrade INFO Selecting v0.7.5 as --from-
122| version from the config.
123| 2025-12-03 23:40:35,757 acapy_agent.commands.upgrade INFO Running upgrade process for
124| v0.8.1
125| 2025-12-03 23:40:35,758 acapy_agent.commands.upgrade INFO No records of <class 'acapy_agent.connections.models.conn_record.ConnRecord'> found
126| 2025-12-03 23:40:35,766 acapy_agent.commands.upgrade INFO acapy_version storage record
127| set to v1.3.2
128| 2025-12-03 23:40:35,769 acapy_agent.core.conductor INFO Listening...

```

```
122| 2025-12-03 23:40:38,661 aiohttp.access INFO 127.0.0.1 [03/Dec/2025:23:40:38 +0000] "GET /status/ready HTTP/1.1" 200 138 "-" "curl/7.88.1"
123| 2025-12-03 23:41:08,696 aiohttp.access INFO 127.0.0.1 [03/Dec/2025:23:41:08 +0000] "GET /status/ready HTTP/1.1" 200 138 "-" "curl/7.88.1"
```

Anhang 5: Summative Evaluation

Anhang 5.1: KRITIS Szenario

Dieses Jupyter Notebook demonstriert einen vollständigen SSI-Workflow für ein KRITIS-Unternehmen im Energiesektor (Bundesministerium des Innern, 2016, §2). Im Fokus stehen die durchgängigen Prozesse der Ausstellung, Verwaltung und Verifikation von Notfall-Wartungszertifikaten für den Zugang zu einem Umspannwerk des Unternehmens auf, welches die Transformation von Hochspannungsenergie in andere Spannungsebenen regelt, Messung und Überwachung des Stromnetzes durchführt sowie die Steuerung der Stromverteilung regelt (Syahputra & Arrozak, 2017, S. 166).

Das zugrundeliegende Szenario basiert auf folgendem Anwendungsfall: Ein Energienetzbetreiber stellt einem Notfalltechniker ein zeitlich begrenztes Zertifikat aus, das den Zugang zu kritischer Infrastruktur, konkret einem Umspannwerk Nord-Ost, gewährt. Das Zutrittssystem verifiziert die Berechtigung des Technikers mittels ZKP, ohne dabei die Identität des Technikers offenzulegen. Dieser Ansatz realisiert das Datenschutzprinzip „Privacy by Design“ gemäß Art. 25 der DSGVO. Die dezentralisierte Natur von SSI-Systemen ermöglicht dabei eine Entkopplung von Identitätsverwaltung und Zutrittskontrolle, wodurch einzelne Kontrollpunkte und zentralisierte Datenbanken als potenzielle Angriffsvektoren reduziert werden.

Anhang 5.1.1: Teil 1: Setup & Verbindungstests

Listing A-33

Jupyter Notebook Cell 1

```
1 # Cell 1: Imports und Konfiguration
2 import requests
3 import json
4 import time
5 import pandas as pd
6 import tabulate
7 from IPython.display import display, Markdown, HTML
8 from datetime import datetime
9 import warnings
10 warnings.filterwarnings('ignore')
11
12 # SSL Warnung für self-signed Zertifikate unterdrücken (PQC Proxy)
13 # import urllib3
14 # urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
15
16 # Agent URLs
17 ISSUER_ADMIN_URL = "https://localhost:8021"
```

```

18 HOLDER_ADMIN_URL = "https://localhost:8031"
19 VERIFIER_ADMIN_URL = "https://localhost:8041"
20 VON_NETWORK_URL = "https://localhost:8000" # PQC HTTPS Reverse Proxy
21
22 # Farben für Visualisierungen
23 ISSUER_COLOR = "#3498db" # Blau
24 HOLDER_COLOR = "#2ecc71" # Grün
25 VERIFIER_COLOR = "#e74c3c" # Rot
26
27 print("Imports erfolgreich")
28 print(f"Issuer: {ISSUER_ADMIN_URL}")
29 print(f"Holder: {HOLDER_ADMIN_URL}")
30 print(f"Verifier: {VERIFIER_ADMIN_URL}")
31 print(f"Ledger: {VON_NETWORK_URL}")
32
33 # =====
34 # Tails-Server Konfiguration (für Revocation)
35 # =====
36 TAILS_SERVER_URL = "https://localhost:6543"
37 TAILS_FILE_COUNT = 100 # Max Credentials per Registry
38
39 print(f"Tails: {TAILS_SERVER_URL}")
40 print(f"Max Credentials per Registry: {TAILS_FILE_COUNT}")
41
42 # =====
43 # Helper Functions
44 # =====
45
46 def api_get(url, path):
47     """GET Request an ACA-Py Admin API"""
48     try:
49         response = requests.get(f"{url}{path}")
50         response.raise_for_status()
51         return response.json()
52     except Exception as e:
53         print(f"GET Fehler: {e}")
54         return None
55
56 def api_post(url, path, data=None):
57     """POST Request an ACA-Py Admin API"""
58     try:
59         response = requests.post(
60             f"{url}{path}",
61             json=data,
62             headers={"Content-Type": "application/json"}
63         )
64         response.raise_for_status()
65         return response.json()
66     except Exception as e:
67         print(f"POST Fehler: {e}")
68         if hasattr(e, 'response') and e.response:
69             print(f"Response: {e.response.text}")
70         return None
71
72 def api_delete(url, path):
73     """DELETE Request an ACA-Py Admin API"""
74     try:

```

```

75         response = requests.delete(
76             f"{url}{path}",
77             headers={"Content-Type": "application/json"}
78         )
79         response.raise_for_status()
80
81     # DELETE kann leeren Body zurückgeben (204 No Content)
82     if response.status_code == 204:
83         return {} # Success, aber kein Content
84
85     # Versuche JSON zu parsen, falls vorhanden
86     try:
87         return response.json()
88     except:
89         return {} # Success, aber kein JSON
90
91     except Exception as e:
92         print(f"DELETE Fehler: {e}")
93         if hasattr(e, 'response') and e.response:
94             print(f"    Response: {e.response.text}")
95         return None
96
97 def pretty_print(data, title=""):
98     """Formatierte JSON-Ausgabe"""
99     if title:
100         print(f"\n{'='*60}")
101         print(f"  {title}")
102         print(f"{'='*60}")
103     print(json.dumps(data, indent=2))
104
105 def wait_for_status(url, path, field, expected_value, timeout=30, interval=1):
106     """Wartet bis ein Feld einen bestimmten Wert hat"""
107     start_time = time.time()
108     while time.time() - start_time < timeout:
109         response = api_get(url, path)
110         if response and response.get(field) == expected_value:
111             return response
112         time.sleep(interval)
113     print(f"Timeout: {field} erreichte nicht '{expected_value}' nach {timeout}s")
114     return None
115
116 def status_badge(status, label="Status"):
117     """HTML Badge für Status"""
118     colors = {
119         "active": "green",
120         "completed": "green",
121         "done": "green",
122         "ready": "green",
123         "pending": "orange",
124         "failed": "red",
125         "error": "red"
126     }
127     color = colors.get(status.lower(), "gray")
128     return f'<span style="background-color:{color};color:white;padding:3px 8px;border-radius:3px;font-weight:bold">{label}: {status}</span>'
129
130 print("\nSetup komplett!")

```

Listing A-34*Jupyter Notebook Cell 1 Output*

```

1 Imports erfolgreich
2 Issuer: https://localhost:8021
3 Holder: https://localhost:8031
4 Verifier: https://localhost:8041
5 Ledger: https://localhost:8000
6 Tails: https://localhost:6543
7 Max Credentials per Registry: 100
8
9 Setup komplett!

```

Listing A-35*Jupyter Notebook Cell 2*

```

1 # Cell 2: Infrastruktur-Status prüfen (Health Check)
2
3 def check_agent_status(name, url):
4     """Prüft ob ein Agent bereit ist"""
5     try:
6         response = api_get(url, "/status/ready")
7         if response and response.get("ready"):
8             return f"{name}: Bereit"
9         else:
10            return f"{name}: Nicht bereit"
11     except:
12         return f"{name}: Nicht erreichbar"
13
14 print("Agenten-Status Check...\n")
15 print(check_agent_status("Issuer", ISSUER_ADMIN_URL))
16 print(check_agent_status("Holder", HOLDER_ADMIN_URL))
17 print(check_agent_status("Verifier", VERIFIER_ADMIN_URL))
18
19 # Detaillierte Informationen mit Conductor-Statistiken
20 print("\nDetaillierte Agent-Informationen:\n")
21 agents_info = []
22 for name, url in [("Issuer", ISSUER_ADMIN_URL), ("Holder", HOLDER_ADMIN_URL), ("Verifier", VERIFIER_ADMIN_URL)]:
23     status = api_get(url, "/status")
24     if status:
25         conductor = status.get("conductor", {})
26         agents_info.append({
27             "Agent": name,
28             "Label": status.get("label", "N/A"),
29             "Version": status.get("version", "N/A"),
30             "In Sessions": conductor.get("in_sessions", 0),
31             "Out Encode": conductor.get("out_encode", 0),
32             "Out Deliver": conductor.get("out_deliver", 0),
33             "Active": conductor.get("task_active", 0),
34             "Done": conductor.get("task_done", 0),
35             "Failed": conductor.get("task_failed", 0),
36             "Pending": conductor.get("task_pending", 0)
37         })

```

```

38
39 df = pd.DataFrame(agents_info)
40 print(df.to_json(orient='records', indent=2))
41
42 print("\nLedger-Verbindung testen...\n")
43
44 try:
45     # Ledger-Status abrufen
46     status_response = requests.get(f"{VON_NETWORK_URL}/status")
47     if status_response.status_code == 200:
48         print("Ledger-Status abrufbar")
49
50     # Genesis-Datei abrufen
51     genesis_response = requests.get(f"{VON_NETWORK_URL}/genesis")
52     if genesis_response.status_code == 200:
53         print(f"Ledger Genesis-Datei ({len(genesis_response.text)}) Bytes) erreichbar\n"
54             "")
55
56 except Exception as e:
57     print(f"Ledger nicht erreichbar: {e}")
58     print("  Bitte starte VON-Network: cd ../von-network && ./manage start --wait\n")
59
60 print(f"\nTails Server Verbindung testen...\n")
61
62 try:
63     # Tails Server Root-Endpoint testen (erwartet 404, da kein Root-Handler)
64     tails_response = requests.get(f"{TAILS_SERVER_URL}")
65
66     if tails_response.status_code == 404:
67         print("Tails Server erreichbar (404 = Normal, kein Root-Endpoint)")
68     elif tails_response.status_code == 200:
69         print(f"Tails Server erreichbar (200 OK)")
70     else:
71         print(f"Tails Server antwortet mit Status {tails_response.status_code}")
72
73     # Alternativer Test: Hash-Endpoint (sollte auch 404 geben ohne gültigen Hash)
74     test_hash = "0" * 64  # Dummy-Hash für Test
75     hash_response = requests.get(f"{TAILS_SERVER_URL}/{test_hash}")
76
77     if hash_response.status_code in [404, 400]:
78         print(f"Tails Server Hash-Endpoint antwortet (Status {hash_response.
79             status_code})")
80
81 except requests.exceptions.ConnectionError as e:
82     print(f"Tails Server nicht erreichbar: {e}")
83     print("  Bitte starte Tails Server:")
84     print("  cd ../indy-tails-server/docker && ./manage start")
85 except Exception as e:
86     print(f" Fehler beim Tails Server Test: {e}")
87
88 # =====
89 # Zeige Ledger-Transaktionen
90 # =====
91 print("\n" + "="*60)
92 print("LEDGER-TRANSAKTIONEN")
93 print("=".*60)

```

```

93| try:
94|     # Hole Domain Ledger Transaktionen
95|     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
96|
97|     if ledger_response.status_code == 200:
98|         ledger_data = ledger_response.json()
99|         ledger_txns = ledger_data.get('results', [])
100|
101    # Filtere nach seqNo 1-5
102    target_seqnos = [1,2,3,4,5]
103    found_txns = []
104|
105    for txn in ledger_txns:
106        txn_metadata = txn.get('txnMetadata', {})
107        seqno = txn_metadata.get('seqNo')
108|
109        if seqno in target_seqnos:
110            found_txns.append(txn)
111|
112    # Sortiere nach seqNo aufsteigend
113    found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
114|
115    if found_txns:
116        print(f"\n{n(len(found_txns)} Transaktionen gefunden:\n")
117|
118        for txn in found_txns:
119            txn_metadata = txn.get('txnMetadata', {})
120            txn_data = txn.get('txn', {})
121            seqno = txn_metadata.get('seqNo')
122            txn_type = txn_data.get('type')
123            txn_time = txn_metadata.get('txnTime', 'N/A')
124|
125            # Typ-Mapping
126            type_names = {
127                '0' : 'NODE (Validator Node Registration)',
128                '1' : 'NYM (DID Registration)',
129                '4' : 'TXN_AUTHOR_AGREEMENT',
130                '5' : 'TXN_AUTHOR_AGREEMENT_AML',
131                '100': 'ATTRIB',
132                '101': 'SCHEMA',
133                '102': 'CRED_DEF',
134                '112': 'CHANGE_KEY',
135                '113': 'REVOC_REG_DEF',
136                '114': 'REVOC_REG_ENTRY',
137                '120': 'AUTH_RULE'
138            }
139            type_name = type_names.get(str(txn_type), f'Type {txn_type}')
140|
141            print(f"====")
142            print(f"Seq No: {seqno} | Time: {txn_time}")
143            print(f"Transaction Type: {type_name}")
144            print(f"====\n")
145|
146    # Zeige vollständige Transaction
147    pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
148    print()

```

```

149
150     else:
151         print(f"\nNYM-Transaktion für DID {did_short} nicht gefunden")
152         print(f"    (Möglicherweise noch nicht im Cache)")
153     else:
154         print(f"\nKonnte Ledger nicht abrufen: {ledger_response.status_code}")
155
156 except Exception as e:
157     print(f"\nFehler beim Abrufen der Ledger-Transaktion: {e}")
158
159 print("=". * 60)

```

Listing A-36*Jupyter Notebook Cell 2 Output*

```

1 Agenten-Status Check...
2
3 Issuer: Bereit
4 Holder: Bereit
5 Verifier: Bereit
6
7 Detaillierte Agent-Informationen:
8
9 [
10   {
11     "Agent": "Issuer",
12     "Label": "Issuer Agent",
13     "Version": "1.3.2",
14     "In Sessions": 0,
15     "Out Encode": 0,
16     "Out Deliver": 0,
17     "Active": 1,
18     "Done": 352,
19     "Failed": 0,
20     "Pending": 0
21   },
22   {
23     "Agent": "Holder",
24     "Label": "Holder Agent",
25     "Version": "1.3.2",
26     "In Sessions": 0,
27     "Out Encode": 0,
28     "Out Deliver": 0,
29     "Active": 1,
30     "Done": 400,
31     "Failed": 1,
32     "Pending": 0
33   },
34   {
35     "Agent": "Verifier",
36     "Label": "Verifier Agent",
37     "Version": "1.3.2",
38     "In Sessions": 0,
39     "Out Encode": 0,
40     "Out Deliver": 0,
41     "Active": 1,

```

```

42      "Done":333,
43      "Failed":0,
44      "Pending":0
45  }
46 ]
47
48 Ledger-Verbindung testen...
49
50 Ledger-Status abrufbar
51 Ledger Genesis-Datei (3099 Bytes) erreichbar
52
53 Tails Server Verbindung testen...
54
55 Tails Server erreichbar (404 = Normal, kein Root-Endpoint)
56 Tails Server Hash-Endpoint antwortet (Status 404)
57
58 =====
59 LEDGER-TRANSAKTIONEN
60 =====
61
62 5 Transaktionen gefunden:
63
64 =====
65 Seq No: 1 | Time: N/A
66 Transaction Type: NYM (DID Registration)
67 =====
68
69
70 =====
71   Ledger Transaction (seqNo 1)
72 =====
73 {
74   "auditPath": [
75     "3XtSy8CQPJUYbc5mFKvUendLZSt4ybG2Y4zRtJEewSL",
76     "96irBGYpWrTvrVATexGGvktPrT3WicixwT8BtoZTtkYX",
77     "Hf2vXibDGJUFB2sMyhEPZZNKEPEiY4iLFxaxckeXwgKx"
78   ],
79   "ledgerSize": 5,
80   "reqSignature": {},
81   "rootHash": "DYLzEift7n9DbiCHqQ5KWrWUPFBizt349popapDfzo5p",
82   "txn": {
83     "data": {
84       "dest": "V4SGRU86Z58d6TV7PBue6f",
85       "role": "0",
86       "verkey": "~CoRER63DVYnWZtK8uAzNbX"
87     },
88     "metadata": {},
89     "type": "1"
90   },
91   "txMetadata": {
92     "seqNo": 1
93   },
94   "ver": "1"
95 }
96
97 =====
98 Seq No: 2 | Time: N/A

```

```

99| Transaction Type: NYM (DID Registration)
100=====
101
102=====
103=====
104| Ledger Transaction (seqNo 2)
105=====
106{
107  "auditPath": [
108    "JBMXqyYxQrtkq9AJjHueEiAtpGiNzVYXkgWqHkZo5zUi",
109    "96irBGYpWrTvrVATexGGvktPrT3WicixwT8BtoZTtkYX",
110    "Hf2vXibDGJUFB2sMyhEPZZNKEPEiY4iLFxaxckeXwgKx"
111  ],
112  "ledgerSize": 5,
113  "reqSignature": {},
114  "rootHash": "DJLzEifT7n9DbiCHqQ5KWrWUPFBiZt349popapDfzo5p",
115  "txn": {
116    "data": {
117      "dest": "Th7MpTaRZVRYnPiabds81Y",
118      "role": "2",
119      "verkey": "~7TYfekw4GUagBnBVCqPjiC"
120    },
121    "metadata": {
122      "from": "V4SGRU86Z58d6TV7PBUE6f"
123    },
124    "type": "1"
125  },
126  "txnMetadata": {
127    "seqNo": 2
128  },
129  "ver": "1"
130}
131=====
132=====
133Seq No: 3 | Time: N/A
134Transaction Type: NYM (DID Registration)
135=====
136
137=====
138=====
139| Ledger Transaction (seqNo 3)
140=====
141{
142  "auditPath": [
143    "3CXeV5MsdAEeU4GLqwGhqvBFuSGAUvqn1xr77rfhWR4e",
144    "D47RQhSxcBgWB1oBZwFeKom3Fvi343NTfpAdKb6uNLuw",
145    "Hf2vXibDGJUFB2sMyhEPZZNKEPEiY4iLFxaxckeXwgKx"
146  ],
147  "ledgerSize": 5,
148  "reqSignature": {},
149  "rootHash": "DJLzEifT7n9DbiCHqQ5KWrWUPFBiZt349popapDfzo5p",
150  "txn": {
151    "data": {
152      "dest": "EbP4aYNNeTHL6q385GuVpRV",
153      "role": "2",
154      "verkey": "~RHGNtfvkgPEUQzQNtNxLNu"
155    },

```

```

156      "metadata": {
157        "from": "V4SGRU86Z58d6TV7PBUE6f"
158      },
159      "type": "1"
160    },
161    "txnMetadata": {
162      "seqNo": 3
163    },
164    "ver": "1"
165  }
166
167=====
168Seq No: 4 | Time: N/A
169Transaction Type: NYM (DID Registration)
170=====
171
172=====
173=====
174  Ledger Transaction (seqNo 4)
175=====
176{
177  "auditPath": [
178    "51ENLADqdvbecXzFQGM5YDk6F9S995TyZfdwFnefKdEb",
179    "D47RQhSxcBgWB1oBZwFeKom3Fvi343NTfpAdKb6uNLuw",
180    "Hf2vXibDGJUFB2sMyhEPZZNKEPEiY4iLFxaxckeXwgKx"
181  ],
182  "ledgerSize": 5,
183  "reqSignature": {},
184  "rootHash": "DJLzEifT7n9DbiCHqQ5KWrWUPFBizt349popapDfzo5p",
185  "txn": {
186    "data": {
187      "dest": "4cU41vWW82ArfxJxHkzXPG",
188      "role": "2",
189      "verkey": "~EMoPA6HrpExVihsVfxD3H"
190    },
191    "metadata": {
192      "from": "V4SGRU86Z58d6TV7PBUE6f"
193    },
194    "type": "1"
195  },
196  "txnMetadata": {
197    "seqNo": 4
198  },
199  "ver": "1"
200}
201
202=====
203Seq No: 5 | Time: N/A
204Transaction Type: NYM (DID Registration)
205=====
206
207=====
208  Ledger Transaction (seqNo 5)
209=====
210{
211  "auditPath": [

```

```

213     "D52hsZf4iH4Kp4x4eEp18FicbPNdirG9TL2cvat1eKvL"
214 ],
215 "ledgerSize": 5,
216 "reqSignature": {},
217 "rootHash": "DJLzEifT7n9DbiCHqQ5KWrWUPFBiZt349popapDfzo5p",
218 "txns": {
219     "data": {
220         "dest": "TWwCRQRZ2ZHMJFn9TzLp7W",
221         "role": "2",
222         "verkey": "~UhP7K35SAXbix1kCQV4Upx"
223     },
224     "metadata": {
225         "from": "V4SGRU86Z58d6TV7PBUE6f"
226     },
227     "type": "1"
228 },
229 "txnsMetadata": {
230     "seqNo": 5
231 },
232 "ver": "1"
233 }
234 =====
235

```

Anhang 5.1.2: Teil 2: DID Setup & Ledger Registration - KRITIS-Identitäten

Listing A-37

Jupyter Notebook Cell 3

```

1 # Cell 3: Issuer - DID erstellen
2
3 print("Issuer: DID erstellen...\n")
4
5 # DID-Erstellungs-Parameter für did:indy
6 # API: POST /did/indy/create
7 did_params = {
8     "options": {
9         "key_type": "ed25519", # Kryptografischer Key-Type (EdDSA)
10        "seed": "00000000000000000000000000000001Issuer01" # 32 Zeichen für reproduzierbare
11          DID
12    }
13 }
14 print(f"DID-Parameter:")
15 print(f"  Method: indy")
16 print(f"  Endpoint: /did/indy/create")
17 print(f"  Key Type: {did_params['options']['key_type']}")
18 print(f"  Seed: {'(zufällig)' if 'seed' not in did_params['options'] else
19       did_params['options']['seed']}")
20 print(f"  Format: did:indy:<identifier>\n")
21 # DID erstellen mit korrektem Endpunkt
22 issuer_did_response = api_post(

```

```

23     ISSUER_ADMIN_URL,
24     "/did/indy/create", # Korrekter Endpunkt für did:indy
25     did_params
26 )
27
28 if issuer_did_response is not None:
29     # Response-Format: {"did": "did:indy:...", "verkey": "..."}
30     issuer_did = issuer_did_response.get("did")
31     issuer_verkey = issuer_did_response.get("verkey")
32
33     print(f"Issuer DID erstellt:")
34     print(f"    DID: {issuer_did}")
35     print(f"    Verkey: {issuer_verkey}")
36
37     # Speichern für spätere Verwendung
38     issuer_info = {
39         "did": issuer_did,
40         "verkey": issuer_verkey,
41         "method": "indy",
42         "key_type": did_params["options"]["key_type"],
43         "role": "ENDORSER"
44     }
45
46     print(f"\nFormat: {issuer_did} (vollständige qualified DID)")
47 else:
48     print("Fehler beim Erstellen der Issuer DID")

```

Listing A-38

Jupyter Notebook Cell 3 Output

```

1 Issuer: DID erstellen...
2
3 DID-Parameter:
4     Method: indy
5     Endpoint: /did/indy/create
6     Key Type: ed25519
7     Seed: 00000000000000000000000000000001Issuer01
8     Format: did:indy:<identifier>
9
10 Issuer DID erstellt:
11     DID: did:indy:9pbXiFBZZGwXKp61HQBz3J
12     Verkey: 2zoa6G7aMfx8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm
13
14 Format: did:indy:9pbXiFBZZGwXKp61HQBz3J (vollständige qualified DID)

```

Listing A-39

Jupyter Notebook Cell 4

```

1 # Cell 4: Issuer DID auf Ledger registrieren
2
3 print("Issuer DID auf Ledger registrieren (via VON-Network)...\\n")
4
5 # Registrierungs-Daten für VON-Network (DID-Based Registration)
6 # Nutzt DID+Verkey aus Cell 3 statt Seed
7 register_data = {

```

```

8     "did": issuer_did,           # DID aus Cell 3 (ACA-Py Wallet)
9     "verkey": issuer_verkey,   # Verkey aus Cell 3 (ACA-Py Wallet)
10    "alias": "Energiennetzbetreiber (Issuer)", 
11    "role": "ENDORSER" # TRUST_ANCHOR oder ENDORSER für Issuer-Rechte
12 }
13
14 print(f"Registrierungs-Parameter:")
15 print(f"  Ledger URL: {VON_NETWORK_URL}/register")
16 print(f"  DID:      {register_data['did']}\"")
17 print(f"  Verkey:   {register_data['verkey'][:20]}...")
18 print(f"  Alias:    {register_data['alias']}\"")
19 print(f"  Role:     {register_data['role']}\\n")
20
21 # Registriere DID auf VON-Network Ledger
22 print(f"POST {VON_NETWORK_URL}/register")
23
24 try:
25     response = requests.post(
26         f"{VON_NETWORK_URL}/register",
27         json=register_data,
28         timeout=10
29     )
30
31     response.raise_for_status()
32
33     # Response format: {"did": "did:indy:...", "verkey": "..."}
34    nym_info = response.json()
35
36     issuer_did = nym_info["did"]
37     issuer_verkey = nym_info["verkey"]
38
39     print(f"Issuer DID auf Ledger registriert")
40     print(f"  DID:      {issuer_did}")
41     print(f"  Verkey:   {issuer_verkey}")
42
43     # Zeige vollständige Ledger-Response
44     pretty_print(nym_info, "VON-Network Registration Response (Issuer)")
45
46     # JETZT: Setze DID als Public DID im ACA-Py Wallet
47     print(f"\nSetze DID als Public DID im Wallet...")
48     public_did_response = api_post(
49         ISSUER_ADMIN_URL,
50         f"/wallet/did/public?did={issuer_did}"
51     )
52
53     if public_did_response is not None:
54         print(f"Issuer DID als Public DID gesetzt")
55     else:
56         print(f"Warnung: Public DID konnte nicht gesetzt werden")
57
58     # Speichern für spätere Verwendung
59     issuer_info = {
60         "did": issuer_did,
61         "verkey": issuer_verkey,
62         "method": "indy",
63         "key_type": "ed25519",
64         "role": "ENDORSER"

```

```

65     }
66
67     # Hole NYM Role vom Ledger zur Verifikation
68     print(f"\nNYM Role vom Ledger verifizieren...")
69     nym_role_response = api_get(
70         ISSUER_ADMIN_URL,
71         f"/ledger/get-nym-role?did={issuer_did}"
72     )
73
74     if nym_role_response is not None:
75         print(f"NYM Role vom Ledger abgerufen")
76         #pretty_print(nym_role_response, "Ledger NYM Role (Issuer DID)")
77
78         # Extrahiere Role
79         role = nym_role_response.get('role', 'N/A')
80         print(f"\nVerifizierte Rolle: {role}")
81     else:
82         print(f"NYM Role konnte nicht abgerufen werden")
83
84     # =====
85     # Zeige Ledger-Transaktionen
86     # =====
87     print("\n" + "="*60)
88     print("LEDGER-TRANSAKTIONEN")
89     print("="*60)
90
91     try:
92         # Hole Domain Ledger Transaktionen
93         ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
94
95         if ledger_response.status_code == 200:
96             ledger_data = ledger_response.json()
97             ledger_txns = ledger_data.get('results', [])
98
99             # Filtere nach seqNo 6 und 7
100            target_seqnos = [6, 7]
101            found_txns = []
102
103            for txn in ledger_txns:
104                txn_metadata = txn.get('txnMetadata', {})
105                seqno = txn_metadata.get('seqNo')
106
107                if seqno in target_seqnos:
108                    found_txns.append(txn)
109
110            # Sortiere nach seqNo aufsteigend
111            found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
112
113            if found_txns:
114                print(f"\n{len(found_txns)} Transaktionen gefunden:\n")
115
116                for txn in found_txns:
117                    txn_metadata = txn.get('txnMetadata', {})
118                    txn_data = txn.get('txn', {})
119                    seqno = txn_metadata.get('seqNo')
120                    txn_type = txn_data.get('type')
121                    txn_time = txn_metadata.get('txnTime', 'N/A')

```

```

122
123     # Typ-Mapping
124     type_names = {
125         '0' : 'NODE (Validator Node Registration)',
126         '1' : 'NYM (DID Registration)',
127         '4' : 'TXN_AUTHOR AGREEMENT',
128         '5' : 'TXN_AUTHOR AGREEMENT_AML',
129         '100': 'ATTRIB',
130         '101': 'SCHEMA',
131         '102': 'CRED_DEF',
132         '112': 'CHANGE_KEY',
133         '113': 'REVOC_REG_DEF',
134         '114': 'REVOC_REG_ENTRY',
135         '120': 'AUTH_RULE'
136     }
137     type_name = type_names.get(str(txn_type), f'Type {txn_type}')
138
139     print(f"
140         =====
141         Seq No: {seqno} | Time: {txn_time}"
142         Transaction Type: {type_name})
143         =====\n")
144
145     # Zeige vollständige Transaction
146     pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
147     print()
148
149     else:
150         print(f"\nNYM-Transaktion für DID {did_short} nicht gefunden")
151         print(f"    (Möglichweise noch nicht im Cache)")
152     else:
153         print(f"\nKonnte Leder nicht abrufen: {ledger_response.status_code}")
154
155     except Exception as e:
156         print(f"\nFehler beim Abrufen der Leder-Transaktion: {e}")
157
158     print("=="*60)
159
160 except requests.exceptions.RequestException as e:
161     print(f" POST Fehler: {e}")
162     print(f"    Stelle sicher dass VON-Network läuft: cd von-network && ./manage start"
163 )

```

Listing A-40

Jupyter Notebook Cell 4 Output

```

1 Issuer DID auf Leder registrieren (via VON-Network)...
2
3 Registrierungs-Parameter:
4     Ledger URL: https://localhost:8000/register
5     DID:          did:indy:9pbXiFBZZGwXKp61HQBz3J
6     Verkey:       2zoa6G7aMfx8GnUEpDxx...
7     Alias:        Energienetzbetreiber (Issuer)
8     Role:         ENDORSER

```

```

9
10 POST https://localhost:8000/register
11 Issuer DID auf Ledger registriert
12     DID: did:indy:9pbXiFBZZGwXKp61HQBz3J
13     Verkey: 2zoa6G7aMfX8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm
14
15 =====
16 VON-Network Registration Response (Issuer)
17 =====
18 {
19     "did": "did:indy:9pbXiFBZZGwXKp61HQBz3J",
20     "seed": null,
21     "verkey": "2zoa6G7aMfX8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm"
22 }
23
24 Setze DID als Public DID im Wallet...
25 Issuer DID als Public DID gesetzt
26
27 NYM Role vom Ledger verifizieren...
28 NYM Role vom Ledger abgerufen
29
30 Verifizierte Rolle: ENDORSER
31
32 =====
33 LEDGER-TRANSAKTIONEN
34 =====
35
36 2 Transaktionen gefunden:
37
38 =====
39 Seq No: 6 | Time: 1765198429
40 Transaction Type: NYM (DID Registration)
41 =====
42
43
44 =====
45     Ledger Transaction (seqNo 6)
46 =====
47 {
48     "auditPath": [
49         "Hf2vXibDGJUFB2sMyhEPZZNKEPEiY4iLFxaxckeXwgKx",
50         "FRQ6sxzHrGmFgEcVDRGZ55wAMJ85fbHBusjMUMrHsJJ8",
51         "D52hsZf4iH4Kp4x4eEpl8FicbPNdirG9TL2cvat1eKvL"
52     ],
53     "ledgerSize": 7,
54     "reqSignature": {
55         "type": "ED25519",
56         "values": [
57             {
58                 "from": "V4SGRU86Z58d6TV7PBue6f",
59                 "value": "CKhwb29Mt4pwW5Cd9jD8gSt3uCCWreuAEm4ypb9iFkh
60                 5i8puahpsoa585fVReDXNP6DzDF5xUF7U4yj4oeSEpjw"
61             }
62         ]
63     },
64     "rootHash": "8mjvdSyKXsKYC39AZEKvEqqdHq87L5alebcPgXUi5jj6",
65     "txns": {

```

```

66      "data": {
67        "alias": "Energienetzbetreiber (Issuer)",
68        "dest": "9pbXiFBZZGwXKp61HQBz3J",
69        "role": "101",
70        "verkey": "2zoa6G7aMfx8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm"
71      },
72      "metadata": {
73        "digest": "6543a880c8311e2a231d588ac1b6a0d53044c51d5c70d15c12b8534af7483815",
74        "from": "V4SGRU86Z58d6TV7PBUE6f",
75        "payloadDigest": "6
76          c05072098125c2479788ca79f7ecf6458efefaf557e31c699d736c7f501e5d5c",
77        "reqId": 1765198429962886624
78      },
79      "protocolVersion": 2,
80      "type": "1"
81    },
82    "txMetadata": {
83      "seqNo": 6,
84      "txId": "fd0d9a9b20213ee708dd54b3129d19875de67c8003f28b7c496eef9786cc176",
85      "txnTime": 1765198429
86    },
87    "ver": "1"
88  }
89 =====
90 Seq No: 7 | Time: 1765198432
91 Transaction Type: ATTRIB
92 =====
93
94 =====
95 =====
96   Ledger Transaction (seqNo 7)
97 =====
98 {
99   "auditPath": [
100     "BXtA9jxNKoyR4nxwq7VQn6L5CFLFGiAHQHEbCfAMCHHW",
101     "D52hsZf4iH4Kp4x4eEp18FicbPNdirG9TL2cvat1eKvL"
102   ],
103   "ledgerSize": 7,
104   "reqSignature": {
105     "type": "ED25519",
106     "values": [
107       {
108         "from": "9pbXiFBZZGwXKp61HQBz3J",
109         "value": "5Yv6acNxZzTpxudAFoU5NrcxWWHy1YEm2Y7myLXF
110         fJxRpDfyPaRjnXoVjdw255BLwJFBJLhzYGUFrcCmKmkmx48"
111       }
112     ]
113   },
114   "rootHash": "8mjvdSyKXsKYC39AZEKvEqqdHq87L5alebcPgXUi5jj6",
115   "tx": {
116     "data": {
117       "dest": "9pbXiFBZZGwXKp61HQBz3J",
118       "raw": "{\"endpoint\":{\"endpoint\":\"https://host.docker.internal:8020\"},\"routingKeys\":[]}"
119     },
120     "metadata": {

```

```

121     "digest": "2268552566b77234a02fd74963edf4fe9146b005c21fc5e04d393b07b2e9511b",
122     "from": "9pbXiFBZZGwXKp61HQBz3J",
123     "payloadDigest": "
124         aa3cb2db276d6d30b8104d631be4a4bac47b8bb2679f0e97105178ec0d828796",
125     "reqId": 1765198430095718777
126 },
127     "protocolVersion": 2,
128     "type": "100"
129 },
130     "txnMetadata": {
131         "seqNo": 7,
132         "txnid": "9pbXiFBZZGwXKp61HQBz3J:1:
133             b6bf7bc8d96f3ea9d132c83b3da8e7760e420138485657372db4d6a981d3fd9e",
134         "txnTime": 1765198432
135     },
136     "ver": "1"
137 }
138 =====
139

```

Listing A-41*Jupyter Notebook Cell 5*

```

1 # Cell 5: Wallet DID Übersicht anzeigen
2
3 print("Wallet-Übersicht - Alle Did's:\n")
4
5 # Alle Did's von allen Agenten abrufen
6 issuer_dids = api_get(ISSUER_ADMIN_URL, "/wallet/did")
7 holder_dids = api_get(HOLDER_ADMIN_URL, "/wallet/did")
8 verifier_dids = api_get(VERIFIER_ADMIN_URL, "/wallet/did")
9
10 #Zeige originale Responses
11 print("Originale /wallet/did Responses:\n")
12 if issuer_dids:
13     pretty_print(issuer_dids, "Issuer Wallet Did's")
14 if holder_dids:
15     pretty_print(holder_dids, "Holder Wallet Did's")
16 if verifier_dids:
17     pretty_print(verifier_dids, "Verifier Wallet Did's")

```

Listing A-42*Jupyter Notebook Cell 5 Output*

```

1 Wallet-Übersicht - Alle Did's:
2
3 Originale /wallet/did Responses:
4
5
6 =====
7     Issuer Wallet Did's
8 =====
9 {
10     "results": [
11         {

```

```

12     "did": "did:indy:9pbXiFBZZGwXKp61HQBz3J",
13     "verkey": "2zoa6G7aMfX8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm",
14     "posture": "posted",
15     "key_type": "ed25519",
16     "method": "indy",
17     "metadata": {
18         "posted": true,
19         "endpoint": "https://host.docker.internal:8020"
20     }
21   ]
22 }
23 =====
24 Holder Wallet DIDs
25 =====
26 {
27   "results": []
28 }
29 =====
30 Verifier Wallet DIDs
31 =====
32 {
33   "results": []
34 }
35 {
36   "results": []
37 }

```

Anhang 5.1.3: Teil 3: Schema & Credential Definition - Notfall-Wartungszertifikat

Listing A-43

Jupyter Notebook Cell 6

```

1 # Cell 6: Schema erstellen (Indy)
2
3 print("Schema erstellen...\n")
4
5 start_time = time.time()
6
7 # Schema-Daten für KRITIS-Notfall-Wartungszertifikat (Indy API)
8 # API: POST /schemas
9 # Format: {"schema_name": "...", "schema_version": "...", "attributes": [...]}
10 schema_data = {
11     "schema_name": "kritis_emergency_maintenance_cert",
12     "schema_version": "1.1", # <-- WICHTIG: Version muss hier angegeben werden!
13     "attributes": [
14         "first_name",
15         "name",
16         "organisation",
17         "role",
18         "cert_type",
19         "facility_type",
20         "epoch_valid_from",
21         "epoch_valid_until",

```

```

22         "security_clearance_level"
23     ]
24 }
25
26 # Schema auf Ledger registrieren (Indy endpoint)
27 schema_response = api_post(
28     ISSUER_ADMIN_URL,
29     "/schemas", # Indy endpoint
30     schema_data
31 )
32
33 if schema_response is not None:
34     # Response Format: {"schema_id": "...", "schema": {...}}
35     schema_id = schema_response.get("schema_id")
36
37     # Korrekte Werte aus Response extrahieren
38     schema_obj = schema_response.get("schema", {})
39     schema_name = schema_obj.get("name", schema_data["schema_name"])
40     schema_version = schema_obj.get("version", schema_data["schema_version"])
41     schema_attributes = schema_obj.get("attrNames", schema_data["attributes"])
42
43     print(f"Schema 'kritis_emergency_maintenance_cert' erstellt (Indy)")
44     print(f"    Schema ID: {schema_id}")
45     print(f"    Endpoint: /schemas")
46     print(f"    Name: {schema_name}")
47     print(f"    Version: {schema_version}")
48     print(f"    Attribute: {''.join(schema_attributes)}\n")
49
50 else:
51     print("Fehler beim Erstellen des Schemas")
52
53 # =====
54 # Zeige Ledger-Transaktionen
55 # =====
56 print("\n" + "="*60)
57 print("LEDGER-TRANSAKTIONEN")
58 print("="*60)
59
60 try:
61     # Hole Domain Ledger Transaktionen
62     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
63
64     if ledger_response.status_code == 200:
65         ledger_data = ledger_response.json()
66         ledger_txns = ledger_data.get('results', [])
67
68         # Filtere nach seqNo 8
69         target_seqnos = [8]
70         found_txns = []
71
72         for txn in ledger_txns:
73             txn_metadata = txn.get('txnMetadata', {})
74             seqno = txn_metadata.get('seqNo')
75
76             if seqno in target_seqnos:
77                 found_txns.append(txn)
78

```

```

79     # Sortiere nach seqNo aufsteigend
80     found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
81
82     if found_txns:
83         print(f"\n{n(len(found_txns))} Transaktionen gefunden:\n")
84
85     for txn in found_txns:
86         txn_metadata = txn.get('txnMetadata', {})
87         txn_data = txn.get('txn', {})
88         seqno = txn_metadata.get('seqNo')
89         txn_type = txn_data.get('type')
90         txn_time = txn_metadata.get('txnTime', 'N/A')
91
92         # Typ-Mapping
93         type_names = {
94             '0': 'NODE (Validator Node Registration)',
95             '1': 'NYM (DID Registration)',
96             '4': 'TXN_AUTHOR AGREEMENT',
97             '5': 'TXN_AUTHOR AGREEMENT_AMI',
98             '100': 'ATTRIB',
99             '101': 'SCHEMA',
100            '102': 'CRED_DEF',
101            '112': 'CHANGE_KEY',
102            '113': 'REVOC_REG_DEF',
103            '114': 'REVOC_REG_ENTRY',
104            '120': 'AUTH_RULE'
105        }
106         type_name = type_names.get(str(txn_type), f'Type {txn_type}')
107
108         print(f"=====")
109         print(f"Seq No: {seqno} | Time: {txn_time}")
110         print(f"Transaction Type: {type_name}")
111         print(f"=====\\n")
112
113         # Zeige vollständige Transaction
114         pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
115         print()
116
117     else:
118         print(f"\nNYM-Transaktion für DID {did_short} nicht gefunden")
119         print(f"    (Möglicherweise noch nicht im Cache)")
120     else:
121         print(f"\nKonnte Leder nicht abrufen: {ledger_response.status_code}")
122
123 except Exception as e:
124     print(f"\nFehler beim Abrufen der Leder-Transaktion: {e}")
125
126 print("=="*60)

```

Listing A-44

Jupyter Notebook Cell 6 Output

```

1 Schema erstellen...
2
3 Schema 'kritis_emergency_maintenance_cert' erstellt (Indy)

```

```

4   Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
5   Endpoint: /schemas
6   Name: kritis_emergency_maintenance_cert
7   Version: 1.1
8   Attribute: security_clearance_level, epoch_valid_from, organisation, facility_type,
9     first_name, epoch_valid_until, name, cert_type, role
10
11 =====
12 LEDGER-TRANSAKTIONEN
13 =====
14
15 1 Transaktionen gefunden:
16
17 =====
18 Seq No: 8 | Time: 1765198929
19 Transaction Type: SCHEMA
20 =====
21
22
23 =====
24   Ledger Transaction (seqNo 8)
25 =====
26 {
27   "auditPath": [
28     "FRQ6sxzHrGmFgEcVDRGZ55wAMJ85fbHBusjMUMrHsJJ8",
29     "BXtA9jxNKoyR4nxwq7VQn6L5CFLFGiAHQHEbCfAMCHHW",
30     "D52hsZf4iH4Kp4x4eEp18FicbPNdirG9TL2cvat1eKvL"
31   ],
32   "ledgerSize": 8,
33   "reqSignature": {
34     "type": "ED25519",
35     "values": [
36       {
37         "from": "9pbXiFBZZGwXKp61HQBz3J",
38         "value": "559SjbmTJSn4f3J46jN4cP9kcWLWygpmnVfms
39           1JF9TCkx4JPkByFdaKqxKPiTxDxtOuemNfPnQdYPMNjTE9"
40       }
41     ]
42   },
43   "rootHash": "3edQ5ymPLwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP",
44   "txns": {
45     "data": {
46       "data": {
47         "attr_names": [
48           "cert_type",
49           "first_name",
50           "organisation",
51           "epoch_valid_until",
52           "role",
53           "epoch_valid_from",
54           "name",
55           "security_clearance_level",
56           "facility_type"
57         ],
58         "name": "kritis_emergency_maintenance_cert",
59         "version": "1.1"
60       }
61     }
62   }
63 }
```

```

60      }
61  },
62  "metadata": {
63    "digest": "4d343a477a3ca559213e5f0b4305db42067ce07bf9f4a1845ae2298874e8b7e7",
64    "from": "9pbXiFBZZGwXKp61HQBz3J",
65    "payloadDigest": "55
66      cbeb77839b6c473e9036784b0b40f0ab7ab0e923b78a60fa441f9224a89322",
67    "reqId": 1765198929055445431
68  },
69  "protocolVersion": 2,
70  "type": "101"
71 },
72 "txnMetadata": {
73   "seqNo": 8,
74   "txnid": "9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1",
75   "txntime": 1765198929
76 },
77 "ver": "1"
78 }
79 =====

```

Listing A-45

Jupyter Notebook Cell 7

```

1 # Cell 7: Credential Definition erstellen (Indy)
2
3 print("Credential Definition erstellen...\n")
4
5 start_time = time.time()
6
7 # Credential Definition Data (Indy API)
8 # API: POST /credential-definitions
9 # Format: {"schema_id": "...", "tag": "...", "support_revocation": bool, ...}
10 cred_def_data = {
11   "schema_id": schema_id, # from Cell 11
12   "tag": "default",
13   "support_revocation": True, # Enable revocation
14   "revocation_registry_size": TAILS_FILE_COUNT # from config
15 }
16
17 print(f"Credential Definition Parameter (Indy):")
18 print(f"  Endpoint:           /credential-definitions")
19 print(f"  Schema ID:         {cred_def_data['schema_id']}")
20 print(f"  Tag:               {cred_def_data['tag']}")
21 print(f"  Support Revocation: {cred_def_data['support_revocation']}")
22 print(f"  RevReg Size:        {cred_def_data['revocation_registry_size']}\n")
23
24 # Create Credential Definition on Ledger (Indy endpoint)
25 cred_def_response = api_post(
26   ISSUER_ADMIN_URL,
27   "/credential-definitions", # Indy endpoint
28   cred_def_data
29 )
30
31 if cred_def_response is not None:

```

```

32     # Response Format: {"credential_definition_id": "..."}
33     cred_def_id = cred_def_response.get("credential_definition_id")
34
35     print(f"Credential Definition erstellt (Indy)")
36     print(f"    Cred Def ID: {cred_def_id}")
37     print(f"    für Schema 'kritis_emergency_maintenance_cert'")
38
39     # Show full Cred Def Response
40     # pretty_print(cred_def_response, "Indy Credential Definition Response (KRITIS)")
41
42 else:
43     print("Fehler beim Erstellen der Credential Definition")
44
45 # =====
46 # Zeige Ledger-Transaktionen
47 # =====
48 print("\n" + "="*60)
49 print("LEDGER-TRANSAKTIONEN")
50 print("="*60)
51
52 try:
53     # Hole Domain Ledger Transaktionen
54     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
55
56     if ledger_response.status_code == 200:
57         ledger_data = ledger_response.json()
58         ledger_txns = ledger_data.get('results', [])
59
60         # Filtere nach seqNo 9-13
61         target_seqnos = [9,10,11,12,13]
62         found_txns = []
63
64         for txn in ledger_txns:
65             txn_metadata = txn.get('txnMetadata', {})
66             seqno = txn_metadata.get('seqNo')
67
68             if seqno in target_seqnos:
69                 found_txns.append(txn)
70
71         # Sortiere nach seqNo aufsteigend
72         found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
73
74     if found_txns:
75         print(f"\n{len(found_txns)} Transaktionen gefunden:\n")
76
77     for txn in found_txns:
78         txn_metadata = txn.get('txnMetadata', {})
79         txn_data = txn.get('txn', {})
80         seqno = txn_metadata.get('seqNo')
81         txn_type = txn_data.get('type')
82         txn_time = txn_metadata.get('txnTime', 'N/A')
83
84         # Typ-Mapping
85         type_names = {
86             '0' : 'NODE (Validator Node Registration)',
87             '1' : 'NYM (DID Registration)',
88             '4' : 'TXN_AUTHOR AGREEMENT',

```

```

89             '5' : 'TXN_AUTHOR AGREEMENT_AML',
90             '100': 'ATTRIB',
91             '101': 'SCHEMA',
92             '102': 'CRED_DEF',
93             '112': 'CHANGE_KEY',
94             '113': 'REVOC_REG_DEF',
95             '114': 'REVOC_REG_ENTRY',
96             '120': 'AUTH_RULE'
97         }
98         type_name = type_names.get(str(txn_type), f'Type {txn_type}')
99
100        print(f"====")
101        print(f"Seq No: {seqno} | Time: {txn_time}")
102        print(f"Transaction Type: {type_name}")
103        print(f"====\n")
104
105        # Zeige vollständige Transaction
106        pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
107        print()
108
109    else:
110        print(f"\nNYM-Transaktion für DID {did_short} nicht gefunden")
111        print(f"  (Möglichlicherweise noch nicht im Cache)")
112    else:
113        print(f"\nKonnte Leder nicht abrufen: {ledger_response.status_code}")
114
115 except Exception as e:
116     print(f"\nFehler beim Abrufen der Leder-Transaktion: {e}")
117
118 print("=="*60)

```

Listing A-46

Jupyter Notebook Cell 7 Output

```

1 Credential Definition erstellen...
2
3 Credential Definition Parameter (Indy):
4   Endpoint:          /credential-definitions
5   Schema ID:        9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
6   Tag:              default
7   Support Revocation: True
8   RevReg Size:      100
9
10 Credential Definition erstellt (Indy)
11   Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
12   für Schema 'kritis_emergency_maintenance_cert'
13
14 =====
15 LEDGER-TRANSAKTIONEN
16 =====
17
18 5 Transaktionen gefunden:
19
20 =====
21 Seq No: 9 | Time: 1765199023

```

```

22| Transaction Type: CRED_DEF
23| =====
24|
25|
26| =====
27| Ledger Transaction (seqNo 9)
28| =====
29{
30  "auditPath": [
31    "HXBvJYp4raWvxc2YPGZi63GAwfXaNfo3tnNMfzd1M1S4",
32    "64KsDF1ChEFno9AQVi6QbzEZ6RNkEJTgtY4SmoSQierj",
33    "5YBFLtPW8BxW8JhBXAct4wLUhP53zefZY3svqsoq12K2",
34    "3edQ5ymPLwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP"
35  ],
36  "ledgerSize": 13,
37  "reqSignature": {
38    "type": "ED25519",
39    "values": [
40      {
41        "from": "9pbXiFBZZGwXKp61HQBz3J",
42        "value": "4TqLMUsejUKG1EADKrcimaTtAxT27pk6Y1
43          vByxWefhbJegk4R9dTZxkGNMa2P98F3y74NB4eDwaQj3pEf
44            vUnbygD"
45      }
46    ],
47  },
48  "rootHash": "41ARQa41wPiJpbwoFr7w6BB3Y66iLWHbmuz84tBEFc
49  "txn": {
50    "data": {
51      "data": {
52        "primary": {
53          "n": "10281269393334237618332478833085297128592880416
54            56445018770380780626224731955086402966982174592411507
55            93759292164843375581672870792507549609025881070059029
56            99569012059221624437458652649731002744035535072320510
57            97344640555305786309596398862760811477348509755096682
58            47559536388723322491759881501464967142625042021950634
59            74590709379823755520735193613328104190124820485724717
60            87489364047623394746143909278968769630514388888301865
61            20447778436083953377999301856507517057286042679094343
62            02052061179063103318370506401665278448306890418334730
63            93881952461509631538635253749457994127628787342472808
64            05983902362485870240931935428443715170617",
65        "r": {
66          "cert_type": "3525685885350642347250119958873755026811
67            522957633518547243819961520253729521705163390662128281
68            100659947377548873990726118958567587241884779850947015
69            446851470518621174242260246721675406567109423807648585
70            318611318059018156362361746259096067266038723705895596
71            093220876250095280642963640906557260166112372721481849
72            431989843865509453854888864631966692285404502280578057
73            721918868775019396126017468321237588005951999342667101
74            975567244876212861888855184811784221600579007710425171
75            580609648567241669678198182453162758482023440157426590
76            771691619172133149688851713491117156155008197673213358
77            875316713959320775727367218011311435",
78        "epoch_valid_from": "38095824218296011101735725593219
79            733656248567836504315284766910602146229358450478114539

```

```

79      823025198949991149752275622931000244720454886784938724
80      905450550272522283065578981652175306387996059549815777
81      727544879899970458395759845156506549712625970899670388
82      294207583619057021604284491705772446291970772201534494
83      508126302873164672075230882118194194482629654114093467
84      700426084200199029421813162173249246860419010465848195
85      910707633747729737872324329237147300460971848764641435
86      740152863156599866188656662672097588486655892715746421
87      143307105793122123488392631257141209577017837760438217
88      94580097290124958486287493355133147337003421",
89      "epoch_valid_until": "14680973432549975411991157440156
90      663509397726968803380045588761027724699355084476591883
91      825212227668972786850841544036009660450745543286264699
92      504034963264074899062628030258629527045078406239725981
93      856646770616598113739789473990417637278478254731994688
94      444076738842870892019139181820137596231408915354234357
95      291843105341408637622019951283953321418913649847371144
96      561934536619979055859996425133654370381497645561219662
97      518390269065734498893970569554769539966440206291530507
98      859191987490188350682362968089839830900632515414715474
99      400865274288497289386900592043713923317231529369332054
100     404647013055752325075709159974250910946425583",
101     "facility_type": "595062717575710994668238940938734807
102     858575150740865779816868809186588085276888258189260457
103     326049240633993158525453995004516858128770077478764760
104     662707092517416575220760082917285325018066989220270025
105     711287446172813428258780648073156550805977523302997930
106     686506926939014462528722726248601504558276505002584521
107     825383154783835244235063460415019367052996073733314190
108     193724578386714250833834627568238450593994554750661863
109     214642667657206798973824064493931047108909153015901843
110     795129099299478918553371157605482045951546345220520602
111     240920027041744720666613936460560291287847757577474685
112     27127573953191722275450519507670497177873",
113     "first_name": "156121461519380424298292418718574326075
114     563477678371398032007947311493792097457113994437563300
115     441410991031578330544123120318869675398369848012575096
116     520704231860406198878736598366391910707963475328379018
117     314895620212026100081783916157950742330812507451654772
118     19523507264610713933385682816746471881809335591903532
119     612594619310336857535281350175402306344253831189945592
120     318245150090617512552186258935479522734234542082002810
121     461645859200972302475885788346000864398002658844962257
122     148412130810688041122837038411449830832849097436729705
123     718116029828253757455923845600999489690503882501644058
124     7864845104128413390680301004111208528",
125     "master_secret": "719150954133244752183971594948903479
126     524386396044792841937602285125175381921943925950159304
127     132642575681280883971815934361686747663056680829710335
128     122664989324241304858850947417770121260898653831591484
129     502093386296347400423866526127863855963487691236562850
130     005973339049657354203655413953447224497040475714353790
131     361203703948242307253413683020978096168369038903109479
132     232973563084115278209020768503212503261781340334895483
133     780220128186741136754232078123977754958748796014376633
134     965241148577842676882513978031731187800872781548919835
135     245340140217768127025745996617405150806112385829127806

```

```

136      79067808174083253800413580518678417409503",
137      "name": "507139475209330359412041956673646670382290502
138      510997416327398817990410291634095878779739727910620391
139      983813348489743782983974205910608455201402883022363030
140      657204230300217006003472034232777595827700699275349376
141      472726421030688458301642805319268862572699616380281698
142      149256210425437977844531965423074013789873092642069860
143      90753",
144      "organisation": "9510167264021885614687405535743007524
145      944893016889819260229094716729897861649969889502212036
146      750229321538650734200306330127232773689469939865899689
147      070492516615728465238729278988718066975226971042518665
148      755377399212641580599260501592205033642808816133021976
149      572658227614869828819166175593661966013773892694492844
150      973525941689522644329814367767130334560379852921542238
151      823637342059481674303290141817243620937852064276411051
152      461547697738554530590173404965337504858475863740776919
153      099876982003005084766640552204421675776338549528241998
154      551725146322875329563300713279344259481053544683702840
155      3340688276237893961510821932537077108757",
156      "role": "395563530099701163940652627405054048882798766
157      671108715408726324441215481630357827076196778750277568
158      825696638995484876172410688858855685967884299877578926
159      313443583940896238443350491578812374333978665353059699
160      559596066426343077451261853141537961994523028629361166
161      627603150886778871652810770119903019741562000480005972
162      417108472300887340241335187817323918093045791638191059
163      822767733463880782634841649824280767803969156144624834
164      311404319066342132889616536476376325317953921950076691
165      046725598460010439417490774915156339698785734342893700
166      483387033327532788710191333995991069122120984481405860
167      68776725335189772775700329133081",
168      "security_clearance_level": "6242343544550071290094928
169      907663448117977302593196784651825226373678939199440163
170      058504073554154551504418212878512647463864530577679028
171      748167713027065237097745671554621508963264018526322739
172      581125596500064997781988824069404787807028616860954572
173      100191611669962108830553784598980577471544467683754087
174      439323758090986602678757392652507765667697136527869728
175      357212776026127704628479865875630145065356272784507801
176      744550339507529178589488999294507749879180468118466591
177      599407835762727403094304443481881622890646807037243874
178      678766414454127205619056010634586741238280596763388454
179      1757073335422180206932822214748818701389818522631849"
180      },
181      "rctxt": "5947310527661084384748647312463192325141246383
182      47616798040672521583722939106309568449503900893285255616
183      50453598319402301909030992815747417296976389802026846055
184      280400772177935138569355196667280507580873412342478848532
185      6387286111841347560302528192995674941761133276177105758
186      74232802793221771136257567273261623626309115862132166003
187      29601336226710484891262929559472657348551152875499590412
188      83480406987986371126564272173879797367626913562453294597
189      38118109227774538122205390834311723056800063504883375713
190      40020297094930055480852885349234400529877113567936709288
191      95651459522039680721583208345252600137326987986347865707
192      80935520753",

```



```

269  "ledgerSize": 13,
270  "reqSignature": {
271    "type": "ED25519",
272    "values": [
273      {
274        "from": "9pbXiFBZZGwXKp61HQBz3J",
275        "value": "522wXmkJ56tLc9zLBLpvjmQrrxXacwcUEmZpo
276        zB1NNveECVAXiYxNV8GcYUJ1yXTKnVEBgUpQmQr5AgfD7S8oH9q"
277      }
278    ]
279  },
280  "rootHash": "41ARQa41wPiJpbwoFr7w6BB3Y66iLWHbmuz84tBEFcnc",
281  "txn": {
282    "data": {
283      "credDefId": "9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default",
284      "id": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:9
285        efc6971-b713-465f-85db-a192b107b73e",
286      "revocDefType": "CL_ACCUM",
287      "tag": "9efc6971-b713-465f-85db-a192b107b73e",
288      "value": {
289        "issuanceType": "ISSUANCE_BY_DEFAULT",
290        "maxCredNum": 100,
291        "publicKeys": {
292          "accumKey": {
293            "z": "1 171D58BB195883545AB3E075FAEB7011618C90C2AB6A2E46E28989FC7D0E46F9 1
294              06EECD5903553820608EAC558B4CD9A8F294EB5C10ED8566A47CB8A8FAD82076 1 1
295              AAB928BF34F09D918D7B1C74A1186F04AE6C94C27D77B94AB4DD276DCDD44EE 1
296              01815895F1C206294EE73CAB992CB672F58157C3F0238382E5B6917C8EEBA021 1 09
297              B5A7ECED3FC84AF833983FB871CA7A46E1F18351AB52F5796BDF455A3BFDC1 1 0
298              F6F11AB712B7192B397A9C4CC93946E2B455DF496F0070BF88B0A9452B4C8C3 1 1
299              AD33C8292498B28A92BD819A0B08BBA8C063B59C83AE972085DE80AD625EA8E 1 225
300              A4616A803A62202CA8191DCFE613BB8C70CC84591AF3271003716770E9A1A 1 22
301              DAE538B68DD1FA88F79BDA08796E5674901D11F9BB187321C95029B9C32C83 1 0
302              FA7E5B2A5AD251AFD24EBD7BE499C102C9ED488E44523FA6E766A3DB6FFB8CE 1 198
303              ED4242B27EA4695A740C0D5DC808C86600A11AE747A4E4DBB2D4F83503AD5 1 1
304              EBEF8A16DF96A7539AAD26BAD9B1FAEAF09ED140BBF0DEEB19592E1B3EDAE81"
305            }
306          },
307        },
308      },
309      "seqNo": 10,

```



```

359      "type": "114"
360    },
361    "txnMetadata": {
362      "seqNo": 11,
363      "txnId": "5:9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:
364      CL_ACCUM:9efc6971-b713-465f-85db-a192b107b73e",
365      "txnTime": 1765199029
366    },
367    "ver": "1"
368  }
369 =====
370 Seq No: 12 | Time: 1765199032
371 Transaction Type: REVOC_REG_DEF
372 =====
373
374 =====
375 =====
376   Ledger Transaction (seqNo 12)
377 =====
378 {
379   "auditPath": [
380     "8YK24RNxLiJUtE7tScUPkuPkZc8UPTyGx7YXV9b2NST2",
381     "5FKBYgm8sbJ8ppfyjrSSpcLeEzZLKDMCWNGuVxp19Ryu",
382     "5YBFLTlPW8BxW8JhBXAct4wLUhP53zefZY3svqsoq12K2",
383     "3edQ5ymPLwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP"
384   ],
385   "ledgerSize": 13,
386   "reqSignature": {
387     "type": "ED25519",
388     "values": [
389       {
390         "from": "9pbXiFBZZGwXKp61HQBz3J",
391         "value": "2eZbXbYtMHW9k58hVgY2xqzsHbYaiGKMTUssNet6Qj
392           Xjk6fYJhmAa2g6sW8aKxSpttmd3PZbjELJxsKZ9J5zqv4N"
393       }
394     ]
395   },
396   "rootHash": "41ARQa41wPiJpbwoFr7w6BB3Y66iLWHbmuz84tBEFcnc",
397   "txns": {
398     "data": {
399       "credDefId": "9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default",
400       "id": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
401           :746cf728-5192-4266-9903-2ffb4b52cc5a",
402       "revocDefType": "CL_ACCUM",
403       "tag": "746cf728-5192-4266-9903-2ffb4b52cc5a",
404       "value": {
405         "issuanceType": "ISSUANCE_BY_DEFAULT",
406         "maxCredNum": 100,
407         "publicKeys": {
408           "accumKey": {
409             "z": "1 073B5A8EC032BBB14E5446AB6212B6A57A47599E9550C70D221F61A7BA624785 1
410               22F3EAE2B892D7F59499DF026AFFD7C993AA6F210F74384330DA134D40CB7FA4 1
411               240218E4070D79AE393B2F99D560A6199DCECB58BB8FDB510A7591F3223B4F82 1 0
412               DA7287AC0CAF306D3D3E2AEA87FBBCDB654B7FC1FB4AA3E2FDD679001E5B688 1 1
413               D305894AC1175B74BF60774DF6B0D0D3214B68075F565FA62992434A7E7CA5A 1 0
414               DBE7E3A553FBDF4CAB921611B04E7B0BEBC53C7F615679FA7992FF89F4E69BF 1 0920
415         }
416       }
417     }
418   }
419 }
```

```

A371E24419599FC419BA7198373719D5AFAA89587BCE0113B707E808209C 1 22
B0BE575646184858C0C0953643F3A222B76C3E556BD26CD4326DC6D3061F85 1 1
F9C5406CCB61F1CF467DA7D8E9071AD5B8761103BFA03058E6955E71A2811BE 1
16736793784BC46DB164979D57393EB5DA106CF5830FA3A09AE2E4E0D7B655E2 1 0
D8AE326C8981107394445B49F2E5B22FB758E74D7DCE5DDCA56A116DFAF8C99 1 10
A48FDC682654E07668738E428E516034EA65A47A398396FD4E2262BB79FCC8"
409     }
410   },
411   "tailsHash": "47xo7WNa2sbxHfnUkenzLgT5LotNci9Z1M5Dsp9vqnAN",
412   "tailsLocation": "https://host.docker.internal:6543/9pbXiFBZZGwXKp61HQBz3J:4:9
    pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:746cf728-5192-4266-9903-2
    ffb4b52cc5a"
413   }
414 },
415   "metadata": {
416     "digest": "fdbcb815ac6ca4c494e5e1f570cf25ea4b50635275b4a84b07c3491964f7025e",
417     "from": "9pbXiFBZZGwXKp61HQBz3J",
418     "payloadDigest": "9
      ca29e738d1adf4584d3595f1e8b0af0baa655158b1cbc8997c2fec3c1e2c8d2",
419     "reqId": 1765199029583049028
420   },
421   "protocolVersion": 2,
422   "type": "113"
423 },
424   "txnMetadata": {
425     "seqNo": 12,
426     "txnId": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
      :746cf728-5192-4266-9903-2ffb4b52cc5a",
427     "txnTime": 1765199032
428   },
429   "ver": "1"
430 }
431 =====
432 Seq No: 13 | Time: 1765199035
433 Transaction Type: REVOC_REG_ENTRY
434 =====
435 =====
436 =====
437 =====
438 =====
439 Ledger Transaction (seqNo 13)
440 =====
441 {
442   "auditPath": [
443     "8vLEbJ5pzVFmhXYiddtoRA8iz7KA8KCANAOKP2Xy2K7q",
444     "3edQ5ymPLwVpBdhaYGETeeewRj1X48jeuZxzHnoDYCgNP"
445   ],
446   "ledgerSize": 13,
447   "reqSignature": {
448     "type": "ED25519",
449     "values": [
450       {
451         "from": "9pbXiFBZZGwXKp61HQBz3J",
452         "value": "2Sx3jiAxTEjgquGZScqZVqJGmmqe8mZnSm4oXsEsMp
          pqj8G7wJoZ2X4zjXMn6qjYKkmxF5SysxhRRCPei6GvXjm"
453       }
454     ]
455   }

```

Anhang 5.1.4: Teil 4: Agent Connections - KRITIS-Akteure

Listing A-47

Jupyter Notebook Cell 8

```
1 # Cell 8: Issuer Discovery - Identifikation via Ledger Identifier
2
3 print("Issuer Discovery - Identifikation vertrauenswürdiger Aussteller")
4 print("="*80)
5 print("KRITIS-Anforderung: Identity Holder müssen vertrauenswürdige Issuers")
6 print("    für gewünschte Credentials entdecken und deren Identität validieren.\n")
7
8 # Phase 1: Ledger-basierte Issuer Discovery
9 print("Phase 1: Ledger-basierte Issuer Discovery\n")
10
11 try:
```

```

12     # Hole Domain Ledger Transaktionen
13     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
14
15     if ledger_response.status_code == 200:
16         ledger_data = ledger_response.json()
17         ledger_txns = ledger_data.get('results', [])
18
19         # Filter: Nur NYM-Transaktionen (DID-Registrierungen, Type '1')
20         nym_txns = [
21             txn for txn in ledger_txns
22             if txn.get('txn', {}).get('type') == '1'
23         ]
24
25         # Filter: Nur Transaktionen mit TRUST_ANCHOR Role (= Issuers)
26         issuer_identities = []
27         for txn in nym_txns:
28             txn_data = txn.get('txn', {}).get('data', {})
29             role = txn_data.get('role')
30
31             # Role '101' = TRUST_ANCHOR (berechtigt, Credentials auszustellen)
32             if role == '101':
33                 full_verkey = txn_data.get('verkey', 'N/A')
34                 issuer_identities.append({
35                     'did': txn_data.get('dest'),
36                     'verkey_full': full_verkey,
37                     'verkey_short': full_verkey[:30] + '...' if len(full_verkey) > 30
38                         else full_verkey,
39                     'alias': txn_data.get('alias', 'N/A'),
40                     'registered_by': txn.get('txn', {}).get('metadata', {}).get('from',
41                         'N/A'),
42                     'txn_time': txn.get('txnMetadata', {}).get('txnTime'),
43                     'seq_no': txn.get('txnMetadata', {}).get('seqNo')
44                 })
45
46             print(f"{len(issuer_identities)} TRUST_ANCHOR Issuer(s) auf dem Ledger
47                   gefunden:\n")
48
49             if issuer_identities:
50                 # Zeige nur relevante Spalten für Identifikation
51                 df_issuers = pd.DataFrame([
52                     'Alias': i['alias'],
53                     'DID': i['did'],
54                     'Verkey (gekürzt)': i['verkey_short'],
55                     'Seq No': i['seq_no']
56                 } for i in issuer_identities])
57                 print(df_issuers.to_json(orient='records', indent=2))
58             else:
59                 print("Keine TRUST_ANCHOR gefunden (Ledger noch leer?)")
60
61         except Exception as e:
62             print(f" Fehler bei Issuer Discovery: {e}")
63             import traceback
64             traceback.print_exc()
65
66 # Phase 2: Issuer Identifier Verification
67 print("\n" * 80)
68 print("Phase 2: Issuer Identifier Verification\n")

```

```

66
67 def verify_issuer_identity(issuer_did, issuer_data, all_txns):
68     """
69     Verifiziert die Identität eines Issuers anhand von Ledger-Daten.
70
71     Identitäts-Eigenschaften:
72     1. DID (Decentralized Identifier)
73     2. Verkey (Public Key für Signaturverifikation)
74     3. TRUST_ANCHOR Role
75     4. Registrierungs-Endorser (Wer hat die Identität bestätigt?)
76     5. On-Ledger Aktivitäten (Schemas, CRED_DEFs, etc.)
77     """
78     identity_info = {
79         'verified': True,
80         'properties': []
81     }
82
83     # 1. DID Identifier
84     identity_info['properties'].append({
85         'property': 'DID (Identifier)',
86         'value': issuer_did,
87         'verified': True
88     })
89
90     # 2. Verkey (Public Key)
91     verkey = issuer_data.get('verkey_full', 'N/A')
92     identity_info['properties'].append({
93         'property': 'Verkey (Public Key)',
94         'value': verkey,
95         'verified': verkey != 'N/A'
96     })
97
98     # 3. TRUST_ANCHOR Role
99     identity_info['properties'].append({
100         'property': 'TRUST_ANCHOR Role',
101         'value': 'Ja (Role 101)',
102         'verified': True
103     })
104
105     # 4. Endorser (Wer hat registriert?)
106     registered_by = issuer_data.get('registered_by', 'N/A')
107     identity_info['properties'].append({
108         'property': 'Registriert von',
109         'value': registered_by,
110         'verified': registered_by != 'N/A'
111     })
112
113     # 5. On-Ledger Aktivitäten
114     activities = []
115
116     # Schemas (Type '101' im Ledger)
117     schemas = [
118         txn for txn in all_txns
119         if txn.get('txns', {}).get('type') == '101'
120         and txn.get('txns', {}).get('metadata', {}).get('from') == issuer_did
121     ]
122     if schemas:

```

```

123     activities.append(f"{len(schemas)} Schema(s)" )
124
125 # Credential Definitions (Type '102')
126 cred_defs = [
127     txn for txn in all_txns
128     if txn.get('txn', {}).get('type') == '102'
129     and txn.get('txn', {}).get('metadata', {}).get('from') == issuer_did
130 ]
131 if cred_defs:
132     activities.append(f"{len(cred_defs)} CRED_DEF(s)" )
133
134 # Revocation Registries (Type '113' oder '114')
135 revoc_regs = [
136     txn for txn in all_txns
137     if txn.get('txn', {}).get('type') in ['113', '114']
138     and txn.get('txn', {}).get('metadata', {}).get('from') == issuer_did
139 ]
140 if revoc_regs:
141     activities.append(f"{len(revoc_regs)} REVOC_REG(s)" )
142
143 activity_summary = ', '.join(activities) if activities else 'Keine'
144 identity_info['properties'].append({
145     'property': 'On-Ledger Aktivitäten',
146     'value': activity_summary,
147     'verified': len(activities) > 0
148 })
149
150 # 6. Zeitstempel (Registrierungszeitpunkt)
151 import datetime
152 txn_time = issuer_data.get('txn_time')
153 if txn_time:
154     registration_date = datetime.datetime.fromtimestamp(txn_time)
155     age_days = (datetime.datetime.now() - registration_date).days
156     identity_info['properties'].append({
157         'property': 'Registriert seit',
158         'value': f'{registration_date.strftime("%Y-%m-%d")} ({age_days} Tage)",
159         'verified': True
160     })
161
162 return identity_info
163
164 # Zeige Identifier-Details für alle gefundenen Issuers
165 if 'issuer_identities' in locals() and issuer_identities:
166     for issuer in issuer_identities[:3]: # Zeige nur erste 3 Issuers
167         print(f"{'='*80}")
168         print(f"Issuer: {issuer['alias']}")"
169         print(f"{'='*80}")
170
171     identity = verify_issuer_identity(issuer['did'], issuer, ledger_txns)
172
173     for prop in identity['properties']:
174         status = "verified" if prop['verified'] else "not verified"
175         value = prop['value']
176         # Kürze lange Werte (z.B. Verkey)
177         if len(str(value)) > 50:
178             value = str(value)[:47] + "..."
179         print(f"    {status} {prop['property'][:25s]} {value}"")

```

```

180
181     print()
182
183 # Phase 3: Issuer-Suche für spezifisches Schema
184 print(f"{'='*80}")
185 print("Phase 3: Issuer-Identifikation für Schema\n")
186
187 # Suche nach dem Schema aus Cell 11
188 desired_schema_name = "kritis_emergency_maintenance_cert"
189
190 try:
191     # Hole alle Schemas vom Ledger (Type '101')
192     schemas = [
193         txn for txn in ledger_txns
194         if txn.get('txn', {}).get('type') == '101'
195     ]
196
197     # Filter nach Schema-Namen
198     matching_schemas = []
199     for schema_txn in schemas:
200         schema_data = schema_txn.get('txn', {}).get('data', {}).get('data', {})
201         schema_name = schema_data.get('name', '')
202
203         if desired_schema_name.lower() in schema_name.lower():
204             schema_version = schema_data.get('version', 'N/A')
205             created_by = schema_txn.get('txn', {}).get('metadata', {}).get('from', 'N/
206                                         A')
207
208             matching_schemas.append({
209                 'schema_name': schema_name,
210                 'schema_version': schema_version,
211                 'schema_id': f'{created_by}:2:{schema_name}:{schema_version}',
212                 'created_by_did': created_by
213             })
214
215 if matching_schemas:
216     print(f"\n{len(matching_schemas)} Schema(s) gefunden für '{desired_schema_name
217     }':\n")
218     df_schemas = pd.DataFrame(matching_schemas)
219     print(df_schemas.to_json(orient='records', indent=2))
220
221     # Identifizierte die Issuers dieser Schemas
222     print(f"\n{'='*80}")
223     print("Issuer-Identifikation für gefundene Schemas:\n")
224
225     for schema in matching_schemas:
226         schema_issuer_did = schema['created_by_did']
227
228         # Finde Issuer-Identität
229         issuer_info = next(
230             (i for i in issuer_identities if i['did'] == schema_issuer_did),
231             None
232         )
233
234         print(f"Schema: {schema['schema_name']} v{schema['schema_version']}")
235         print(f"{'-'*80}")

```

```

235     if issuer_info:
236         # Zeige Issuer-Identifikation
237         print(f"Issuer identifiziert:")
238         print(f"    Alias: {issuer_info['alias']}")
239         print(f"    DID: {issuer_info['did']}")
240         print(f"    Verkey: {issuer_info['verkey_short']}")
241         print(f"    TRUST_ANCHOR: Ja (Role 101)")
242         print(f"    Ledger Seq No: {issuer_info['seq_no']}")

243
244         # Zeige vollständige Identitäts-Verifikation
245         print(f"\nVollständige Identitäts-Verifikation:")
246         identity = verify_issuer_identity(schema_issuer_did, issuer_info,
247                                           ledger_txns)
248         for prop in identity['properties']:
249             status = "verified" if prop['verified'] else "not verified"
250             value = prop['value']
251             if len(str(value)) > 40:
252                 value = str(value)[:37] + "..."
253             print(f"    {status} {prop['property'][:20s]} {value}")

254         print(f"\nIssuer erfolgreich über DID identifiziert und verifiziert")
255     else:
256         print(f"Issuer NICHT als TRUST_ANCHOR registriert")
257         print(f"    DID: {schema_issuer_did}")
258         print(f"    Status: Keine TRUST_ANCHOR Role auf dem Ledger")

259
260         print(f"\n{'='*80}\n")
261     else:
262         print(f"Keine Schemas gefunden für '{desired_schema_name}'")
263         print(f"Tip: Stelle sicher, dass Cell 11 (Schema erstellen) ausgeführt wurde")
264
265 except Exception as e:
266     print(f"Fehler bei Schema-Suche: {e}")
267     import traceback
268     traceback.print_exc()

269
270 print(f"\n{'='*80}")
271 print("Issuer Identifier-basierte Discovery:")
272 print("    1. DID (Decentralized Identifier) - Eindeutiger Identifier")
273 print("    2. Verkey (Public Key) - Kryptografische Identität")
274 print("    3. TRUST_ANCHOR Role - Berechtigung zum Ausstellen")
275 print("    4. Ledger Endorsement - Wer hat die Identität bestätigt?")
276 print("    5. On-Ledger Aktivitäten - Schemas, CRED_DEFs, REVOC_REGs")
277 print(f"{'='*80}")
278 print("Identifikations-Methode:")
279 print("    Der Issuer wird über seinen DID (Decentralized Identifier) eindeutig")
280 print("    identifiziert. Der DID ist im Schema-Identifier eingebettet:")
281 print("    Format: <issuer_did>:<schema_name>:<version>")
282 print("    ")
283 print("    Die Vertrauenswürdigkeit wird durch die TRUST_ANCHOR Role auf dem")
284 print("    Hyperledger Indy Ledger garantiert (Byzantine Fault Tolerant).")
285 print(f"{'='*80}")

```

Listing A-48*Jupyter Notebook Cell 8 Output*

```

1 Issuer Discovery - Identifikation vertrauenswürdiger Aussteller
2 =====
3 KRITIS-Anforderung: Identity Holder müssen vertrauenswürdige Issuers
4     für gewünschte Credentials entdecken und deren Identität validieren.
5
6 Phase 1: Ledger-basierte Issuer Discovery
7
8 1 TRUST_ANCHOR Issuer(s) auf dem Ledger gefunden:
9
10 [
11     {
12         "Alias": "Energienetzbetreiber (Issuer)",
13         "DID": "9pbXiFBZZGwXKp61HQBz3J",
14         "Verkey (gek\u00fcrczt)": "2zoa6G7aMfx8GnUEpDxxunFHE7fZkt...",
15         "Seq No": 6
16     }
17 ]
18
19 =====
20 Phase 2: Issuer Identifier Verification
21
22 =====
23 Issuer: Energienetzbetreiber (Issuer)
24 =====
25 DID (Identifier)          9pbXiFBZZGwXKp61HQBz3J
26 Verkey (Public Key)       2zoa6G7aMfx8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm
27 TRUST_ANCHOR Role        Ja (Role 101)
28 Registriert von           V4SGRU86Z58d6TV7PBUs6f
29 On-Ledger Aktivitäten    1 Schema(s), 1 CRED_DEF(s), 4 REVOC_REG(s)
30 Registriert seit          2025-12-08 (0 Tage)
31
32 =====
33 Phase 3: Issuer-Identifikation für Schema
34
35 1 Schema(s) gefunden für 'kritis_emergency_maintenance_cert':
36
37 [
38     {
39         "schema_name": "kritis_emergency_maintenance_cert",
40         "schema_version": "1.1",
41         "schema_id": "9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1",
42         "created_by_did": "9pbXiFBZZGwXKp61HQBz3J"
43     }
44 ]
45
46 =====
47 Issuer-Identifikation für gefundene Schemas:
48
49 Schema: kritis_emergency_maintenance_cert v1.1
50
51 Issuer identifiziert:
52     Alias:      Energienetzbetreiber (Issuer)
53     DID:        9pbXiFBZZGwXKp61HQBz3J
54     Verkey:     2zoa6G7aMfx8GnUEpDxxunFHE7fZkt...

```

```
55 TRUST_ANCHOR: Ja (Role 101)
56 Ledger Seq No: 6
57
58 Vollständige Identitäts-Verifikation:
59 DID (Identifier) 9pbXiFBZZGwXKp61HQBz3J
60 Verkey (Public Key) 2zoa6G7aMfX8GnUEpDxxunFHE7fZktRiiHk1v...
61 TRUST_ANCHOR Role Ja (Role 101)
62 Registriert von V4SGRU86Z58d6TV7PBUE6f
63 On-Ledger Aktivitäten 1 Schema(s), 1 CRED_DEF(s), 4 REVOC_R...
64 Registriert seit 2025-12-08 (0 Tage)
65
66 Issuer erfolgreich über DID identifiziert und verifiziert
67
68 =====
69
70
71 =====
72 Issuer Identifier-basierte Discovery:
73     1. DID (Decentralized Identifier) - Eindeutiger Identifier
74     2. Verkey (Public Key) - Kryptografische Identität
75     3. TRUST_ANCHOR Role - Berechtigung zum Ausstellen
76     4. Ledger Endorsement - Wer hat die Identität bestätigt?
77     5. On-Ledger Aktivitäten - Schemas, CRED_DEFS, REVOC_REGS
78 =====
79 Identifikations-Methode:
80     Der Issuer wird über seinen DID (Decentralized Identifier) eindeutig
81     identifiziert. Der DID ist im Schema-Identifier eingebettet:
82     Format: <issuer_did>:2:<schema_name>:<version>
83
84     Die Vertrauenswürdigkeit wird durch die TRUST_ANCHOR Role auf dem
85     Hyperledger Indy Ledger garantiert (Byzantine Fault Tolerant).
86 =====
```

Listing A-49

Jupyter Notebook Cell 9

```
1 # Cell 9: Connection Issuer --> Holder erstellen
2
3 print("Connection: Issuer --> Holder erstellen...\n")
4
5 start_time = time.time()
6
7 # =====
8 # 1. PRE-CHECK: Existierende Connections prüfen
9 # =====
10 print("Pre-Check: Prüfe existierende Connections...\n")
11
12 existing_conn_issuer_holder = None
13 existing_conn_holder = None
14
15 # Prüfe Issuer-Seite: GET /connections
16 issuer_conns_response = api_get(ISSUER_ADMIN_URL, "/connections")
17 if issuer_conns_response and "results" in issuer_conns_response:
18     print(f"    Issuer: {len(issuer_conns_response['results'])} Connection(s) gefunden")
19     for conn in issuer_conns_response["results"]:
```

```

20         if conn.get("state") == "completed":
21             existing_conn_issuer_holder = conn["connection_id"]
22             print(f"Connection {existing_conn_issuer_holder} (State: completed)")
23             break
24     else:
25         print("    Issuer: Keine Connections gefunden")
26
27 # Prüfe Holder-Seite: GET /connections
28 holder_conns_response = api_get(HOLDER_ADMIN_URL, "/connections")
29 if holder_conns_response and "results" in holder_conns_response:
30     print(f"    Holder: {len(holder_conns_response['results'])} Connection(s) gefunden")
31     for conn in holder_conns_response["results"]:
32         if conn.get("state") == "completed":
33             existing_conn_holder = conn["connection_id"]
34             print(f"Connection {existing_conn_holder} (State: completed)")
35             break
36     else:
37         print("    Holder: Keine Connections gefunden")
38
39 if existing_conn_issuer_holder and existing_conn_holder:
40     print("\nExistierende Connection gefunden!")
41     print(f"    Issuer Connection ID: {existing_conn_issuer_holder}")
42     print(f"    Holder Connection ID: {existing_conn_holder}")
43     print("    Überspringe Connection-Erstellung\n")
44     conn_id_issuer_holder = existing_conn_issuer_holder
45     conn_id_holder = existing_conn_holder
46 else:
47     print("\n    Keine vollständige existierende Connection gefunden")
48     print("    Erstelle neue Connection mit did:peer:4...\n")
49
50 # =====
51 # 2. HAUPT-WORKFLOW: Connection mit did:peer erstellen
52 # =====
53
54 # Initialisiere Variablen
55 conn_id_issuer_holder = None
56 conn_id_holder = None
57 invitation_msg_id = None
58 invitation_key = None
59
60 # Issuer erstellt Out-of-Band Invitation mit did:peer:4
61 invitation_data = {
62     "handshake_protocols": ["https://didcomm.org/didexchange/1.1"],
63     "use_did_method": "did:peer:4",
64     "my_label": "Issuer Agent",
65     "goal": "Establish connection for credential issuance",
66     "goal_code": "issue-vc",
67     "accept": [
68         "didcomm/aip1",
69         "didcomm/aip2;env=rfc19"
70     ]
71 }
72
73 print("Invitation-Modus: did:peer:4 (P2P Connection)")
74
75 # API Call mit Query-Parametern für auto-accept

```

```

76     invitation_response = api_post(
77         ISSUER_ADMIN_URL,
78         "/out-of-band/create-invitation?auto_accept=true&multi_use=false",
79         invitation_data
80     )
81
82     if invitation_response and "invitation" in invitation_response:
83         invitation = invitation_response["invitation"]
84         oob_id = invitation_response.get("oob_id")
85         invitation_url = invitation_response.get("invitation_url", "")
86         state = invitation_response.get("state")
87
88         # WICHTIG: Speichere invitation_msg_id aus der Invitation
89         invitation_msg_id = invitation.get("@id")
90
91         # Optional: Extrahiere invitation_key aus services (falls vorhanden)
92         services = invitation.get("services", [])
93         if services and len(services) > 0:
94             # Bei did:peer:4 ist der Service oft inline
95             service = services[0]
96             if isinstance(service, dict):
97                 recipient_keys = service.get("recipientKeys", [])
98                 if recipient_keys:
99                     invitation_key = recipient_keys[0]
100
101            print(f"Issuer: Invitation erstellt")
102            print(f"    OOB ID: {oob_id}")
103            print(f"    State: {state}")
104            print(f"    Invitation Msg ID: {invitation_msg_id}")
105            if invitation_key:
106                print(f"    Invitation Key: {invitation_key[:30]}...")
107            print(f"    Services: {len(invitation.get('services', []))} service(s)")
108            print(f"    Label: {invitation_data['my_label']}")
109            print(f"    Goal: {invitation_data['goal']}")
110
111            # Holder akzeptiert Invitation (mit auto-accept)
112            receive_response = api_post(
113                HOLDER_ADMIN_URL,
114                "/out-of-band/receive-invitation?auto_accept=true",
115                invitation
116            )
117
118            if receive_response is not None:
119                conn_id_holder = receive_response.get("connection_id")
120                print(f"\nHolder: Invitation akzeptiert")
121                print(f"    Connection ID (Holder): {conn_id_holder}")
122
123                # Issuer findet seine Connection via invitation_msg_id
124                print(f"\n    Issuer sucht Connection via invitation_msg_id...")
125                time.sleep(2)  # Kurze Wartezeit für DIDExchange Protocol
126
127                # Hole alle Issuer Connections
128                issuer_conns = api_get(ISSUER_ADMIN_URL, "/connections")
129                if issuer_conns and "results" in issuer_conns:
130                    connections = issuer_conns["results"]
131
132                    print(f"    Prüfe {len(connections)} Connection(s)...")

```

```

133
134     # Finde Connection anhand invitation_msg_id (exaktes Match!)
135     connection_found = False
136     for conn in connections:
137         conn_invitation_msg_id = conn.get("invitation_msg_id")
138         conn_invitation_key = conn.get("invitation_key")
139
140         # Match via invitation_msg_id (bevorzugt)
141         if invitation_msg_id and conn_invitation_msg_id == invitation_msg_id:
142             conn_id_issuer_holder = conn["connection_id"]
143             conn_state = conn.get("state")
144             print(f"\nIssuer: Connection gefunden via invitation_msg_id!")
145             print(f"Connection ID: {conn_id_issuer_holder}")
146             print(f"State: {conn_state}")
147             print(f"Invitation Msg ID: {conn_invitation_msg_id}")
148             print(f"Their Label: {conn.get('their_label', 'N/A')}")
149             print(f"Updated At: {conn.get('updated_at', 'N/A')}")
150             connection_found = True
151             break
152
153         # Fallback: Match via invitation_key
154         elif invitation_key and conn_invitation_key == invitation_key:
155             conn_id_issuer_holder = conn["connection_id"]
156             conn_state = conn.get("state")
157             print(f"\nIssuer: Connection gefunden via invitation_key!")
158             print(f"Connection ID: {conn_id_issuer_holder}")
159             print(f"State: {conn_state}")
160             print(f"Invitation Key: {conn_invitation_key[:30]}...")
161             print(f"Their Label: {conn.get('their_label', 'N/A')}")
162             connection_found = True
163             break
164
165         if not connection_found:
166             print(f"Keine Connection mit matching invitation_msg_id gefunden")
167             print(f"Gesuchte invitation_msg_id: {invitation_msg_id}")
168         else:
169             print(f"Konnte Issuer Connections nicht abrufen")
170         else:
171             print("Holder konnte Invitation nicht akzeptieren")
172     else:
173         print("Fehler beim Erstellen der Invitation")
174
175 # =====
176 # 3. POST-VALIDATION: Connection-Status verifizieren
177 # =====
178 print("\nPost-Validation: Verifizierte Connection-Status...\n")
179
180 validation_success = False
181 issuer_state = None
182 holder_state = None
183
184 if conn_id_issuer_holder:
185     # Prüfe Issuer-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
186     issuer_conn_detail = api_get(
187         ISSUER_ADMIN_URL,
188         f"/connections/{conn_id_issuer_holder}"

```

```

189     )
190
191     if issuer_conn_detail:
192         issuer_state = issuer_conn_detail.get("state")
193         issuer_their_label = issuer_conn_detail.get("their_label")
194         issuer_their_role = issuer_conn_detail.get("their_role")
195         issuer_invitation_msg_id = issuer_conn_detail.get("invitation_msg_id")
196
197         print(f"\nIssuer Connection {conn_id_issuer}:")
198         print(f"State: {issuer_state}")
199         print(f"Their Label: {issuer_their_label}")
200         print(f"Their Role: {issuer_their_role}")
201         print(f"Invitation Msg ID: {issuer_invitation_msg_id}")
202
203 # Prüfe Holder-Seite (falls conn_id_holder bekannt)
204 if conn_id_holder:
205     # Prüfe Holder-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
206     holder_conn_detail = api_get(
207         HOLDER_ADMIN_URL,
208         f"/connections/{conn_id_holder}"
209     )
210
211     if holder_conn_detail:
212         holder_state = holder_conn_detail.get("state")
213         holder_their_label = holder_conn_detail.get("their_label")
214         holder_their_role = holder_conn_detail.get("their_role")
215         holder_invitation_msg_id = holder_conn_detail.get("invitation_msg_id")
216
217         print(f"\nHolder Connection {conn_id_holder}:")
218         print(f"State: {holder_state}")
219         print(f"Their Label: {holder_their_label}")
220         print(f"Their Role: {holder_their_role}")
221         print(f"Invitation Msg ID: {holder_invitation_msg_id}")
222
223     # Verifiziere dass invitation_msg_id übereinstimmt
224     if invitation_msg_id and issuer_invitation_msg_id == holder_invitation_msg_id:
225         print(f"\nInvitation Msg IDs stimmen überein!")
226
227 # Validierung
228 if issuer_state == "active" and holder_state == "active":
229     validation_success = True
230 elif issuer_state == "active" and not conn_id_holder:
231     # Bei existierender Connection nur Issuer-Seite bekannt
232     validation_success = True
233
234 # Finale Ausgabe
235 print(f"\n{'='*60}")
236 if validation_success:
237     print(f"\nCONNECTION ESTABLISHED (did:peer:4): Issuer <--> Holder")
238     print(f"    Issuer Connection ID: {conn_id_issuer_holder}")
239     if conn_id_holder:
240         print(f"        Holder Connection ID: {conn_id_holder}")
241     print(f"    Issuer State: {issuer_state}")
242     if holder_state:
243         print(f"        Holder State: {holder_state}")
244     if invitation_msg_id:
245         print(f"        Invitation Msg ID: {invitation_msg_id}")

```

```

246 | else:
247 |     print(f"CONNECTION STATUS UNCLEAR")
248 |     print(f"    Issuer State: {issuer_state if issuer_state else 'N/A'}")
249 |     print(f"    Holder State: {holder_state if holder_state else 'N/A'}")
250 |     if conn_id_issuer_holder:
251 |         print(f"    Issuer Connection ID: {conn_id_issuer_holder}")
252 |     if conn_id_holder:
253 |         print(f"    Holder Connection ID: {conn_id_holder}")
254 |
255 | print(f"{'='*60}")

```

Listing A-50

Jupyter Notebook Cell 9 Output

```

1 Connection: Issuer --> Holder erstellen...
2
3 Pre-Check: Prüfe existierende Connections...
4
5     Issuer: 0 Connection(s) gefunden
6     Holder: 0 Connection(s) gefunden
7
8     Keine vollständige existierende Connection gefunden
9     Erstelle neue Connection mit did:peer:4...
10
11 Invitation-Modus: did:peer:4 (P2P Connection)
12 Issuer: Invitation erstellt
13     OOB ID: 3b5fe1c5-a9b0-46d1-8cd9-0ead4720f4aa
14     State: initial
15     Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
16     Services: 1 service(s)
17     Label: Issuer Agent
18     Goal: Establish connection for credential issuance
19
20 Holder: Invitation akzeptiert
21     Connection ID (Holder): 860434c0-7632-41d9-90ea-addaafa68a33
22
23 Issuer sucht Connection via invitation_msg_id...
24     Prüfe 1 Connection(s)...
25
26 Issuer: Connection gefunden via invitation_msg_id!
27     Connection ID: c5399b17-85f2-4adf-99a5-5aeeee8647d9
28     State: active
29     Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
30     Their Label: Holder Agent
31     Updated At: 2025-12-08T13:27:30.289055Z
32
33 Post-Validation: Verifizierte Connection-Status...
34
35     Issuer Connection c5399b17-85f2-4adf-99a5-5aeeee8647d9:
36         State: active
37         Their Label: Holder Agent
38         Their Role: invitee
39         Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
40
41     Holder Connection 860434c0-7632-41d9-90ea-addaafa68a33:
42         State: active

```

```

43     Their Label: Issuer Agent
44     Their Role: inviter
45     Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
46     Invitation Msg IDs stimmen überein!
47
48 =====
49 CONNECTION ESTABLISHED (did:peer:4): Issuer <--> Holder
50     Issuer Connection ID: c5399b17-85f2-4adf-99a5-5aeeee8647d9
51     Holder Connection ID: 860434c0-7632-41d9-90ea-addaafa68a33
52     Issuer State: active
53     Holder State: active
54     Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
55 =====

```

Listing A-51

Jupyter Notebook Cell 10

```

1 # Cell 10: Connection Holder --> Verifier erstellen
2
3 print("Connection: Holder --> Verifier erstellen...\n")
4
5 start_time = time.time()
6
7 # =====
8 # 1. PRE-CHECK: Existierende Connections prüfen
9 # =====
10 print("Pre-Check: Prüfe existierende Connections...\n")
11
12 existing_conn_verifier_holder = None
13 existing_conn_holder_verifier = None
14
15 # Prüfe Verifier-Seite: GET /connections
16 verifier_conns_response = api_get(VERIFIER_ADMIN_URL, "/connections")
17 if verifier_conns_response and "results" in verifier_conns_response:
18     print(f"    Verifier: {len(verifier_conns_response['results'])} Connection(s) gefunden")
19     for conn in verifier_conns_response["results"]:
20         if conn.get("state") == "completed":
21             existing_conn_verifier_holder = conn["connection_id"]
22             print(f"        --> Connection {existing_conn_verifier_holder} (State: completed)")
23             break
24     else:
25         print("    Verifier: Keine Connections gefunden")
26
27 # Prüfe Holder-Seite: GET /connections (für Holder --> Verifier)
28 holder_conns_response = api_get(HOLDER_ADMIN_URL, "/connections")
29 if holder_conns_response and "results" in holder_conns_response:
30     print(f"    Holder: {len(holder_conns_response['results'])} Connection(s) gefunden")
31     #
32     # Finde Connection zum Verifier (basierend auf Label oder neueste)
33     holder_verifier_conns = [c for c in holder_conns_response["results"] if c.get("their_label") == "Verifier Agent"]
34     if holder_verifier_conns:
35         for conn in holder_verifier_conns:
36             if conn.get("state") == "completed":

```

```

36             existing_conn_holder_verifier = conn["connection_id"]
37             print(f"      --> Connection {existing_conn_holder_verifier} (State:
38                   completed, Partner: Verifier)")
39             break
40         else:
41             print("      Holder: Keine Connections gefunden")
42
42 if existing_conn_verifier_holder and existing_conn_holder_verifier:
43     print(f"\nExistierende Connection gefunden!")
44     print(f"      Verifier Connection ID: {existing_conn_verifier_holder}")
45     print(f"      Holder Connection ID: {existing_conn_holder_verifier}")
46     print("      Überspringe Connection-Erstellung\n")
47     conn_id_verifier_holder = existing_conn_verifier_holder
48     conn_id_holder_verifier = existing_conn_holder_verifier
49 else:
50     print("\n      Keine vollständige existierende Connection gefunden")
51     print("      Erstelle neue Connection mit did:peer:4...\n")
52
53 # =====
54 # 2. HAUPT-WORKFLOW: Connection mit did:peer erstellen
55 # =====
56
57 # Initialisiere Variablen
58 conn_id_verifier_holder = None
59 conn_id_holder_verifier = None
60 invitation_msg_id = None
61 invitation_key = None
62
63 # Verifier erstellt Out-of-Band Invitation mit did:peer:4
64 invitation_data = {
65     "handshake_protocols": ["https://didcomm.org/didexchange/1.1"],
66     "use_did_method": "did:peer:4",
67     "my_label": "Verifier Agent",
68     "goal": "Establish connection for credential verification",
69     "goal_code": "verify-vc",
70     "accept": [
71         "didcomm/aip1",
72         "didcomm/aip2;env=rfc19"
73     ]
74 }
75
76 print("Invitation-Modus: did:peer:4 (P2P Connection)")
77
78 # API Call mit Query-Parametern für auto-accept
79 invitation_response = api_post(
80     VERIFIER_ADMIN_URL,
81     "/out-of-band/create-invitation?auto_accept=true&multi_use=false",
82     invitation_data
83 )
84
85 if invitation_response and "invitation" in invitation_response:
86     invitation = invitation_response["invitation"]
87     oob_id = invitation_response.get("oob_id")
88     invitation_url = invitation_response.get("invitation_url", "")
89     state = invitation_response.get("state")
90
91     # WICHTIG: Speichere invitation_msg_id aus der Invitation

```

```

92     invitation_msg_id = invitation.get("@id")
93
94     # Optional: Extrahiere invitation_key aus services (falls vorhanden)
95     services = invitation.get("services", [])
96     if services and len(services) > 0:
97         service = services[0]
98         if isinstance(service, dict):
99             recipient_keys = service.get("recipientKeys", [])
100            if recipient_keys:
101                invitation_key = recipient_keys[0]
102
103            print(f"Verifier: Invitation erstellt")
104            print(f"    OOB ID: {oob_id}")
105            print(f"    State: {state}")
106            print(f"    Invitation Msg ID: {invitation_msg_id}")
107            if invitation_key:
108                print(f"    Invitation Key: {invitation_key}")
109            print(f"    Services: {len(invitation.get('services', []))} service(s)")
110            print(f"    Label: {invitation_data['my_label']}")
111            print(f"    Goal: {invitation_data['goal']}")
112
113        # Holder akzeptiert Invitation (mit auto-accept)
114        receive_response = api_post(
115            HOLDER_ADMIN_URL,
116            "/out-of-band/receive-invitation?auto_accept=true",
117            invitation
118        )
119
120        if receive_response is not None:
121            conn_id_holder_verifier = receive_response.get("connection_id")
122            print(f"\nHolder: Invitation akzeptiert")
123            print(f"    Connection ID (Holder): {conn_id_holder_verifier}")
124
125        # Verifier findet seine Connection via invitation_msg_id
126        print(f"\n    Verifier sucht Connection via invitation_msg_id...")
127        time.sleep(2)  # Kurze Wartezeit für DIDExchange Protocol
128
129        # Hole alle Verifier Connections
130        verifier_conns = api_get(VERIFIER_ADMIN_URL, "/connections")
131        if verifier_conns and "results" in verifier_conns:
132            connections = verifier_conns["results"]
133
134            print(f"    Prüfe {len(connections)} Connection(s)...")
135
136            # Finde Connection anhand invitation_msg_id (exaktes Match!)
137            connection_found = False
138            for conn in connections:
139                conn_invitation_msg_id = conn.get("invitation_msg_id")
140                conn_invitation_key = conn.get("invitation_key")
141
142                # Match via invitation_msg_id (bevorzugt)
143                if invitation_msg_id and conn_invitation_msg_id ==
144                    invitation_msg_id:
145                    conn_id_verifier_holder = conn["connection_id"]
146                    conn_state = conn.get("state")
147                    print(f"\n    Verifier: Connection gefunden via
148          invitation_msg_id!")

```

```

147             print(f"      Connection ID: {conn_id_verifier_holder}")
148             print(f"      State: {conn_state}")
149             print(f"      Invitation Msg ID: {conn_invitation_msg_id}")
150             print(f"      Their Label: {conn.get('their_label', 'N/A')}")
151             print(f"      Updated At: {conn.get('updated_at', 'N/A')}")
152             connection_found = True
153             break
154
155         # Fallback: Match via invitation_key
156         elif invitation_key and conn_invitation_key == invitation_key:
157             conn_id_verifier_holder = conn["connection_id"]
158             conn_state = conn.get("state")
159             print(f"\n      Verifier: Connection gefunden via invitation_key!
160                  ")
160             print(f"      Connection ID: {conn_id_verifier_holder}")
161             print(f"      State: {conn_state}")
162             print(f"      Invitation Key: {conn_invitation_key}...")
163             print(f"      Their Label: {conn.get('their_label', 'N/A')}")
164             connection_found = True
165             break
166
167         if not connection_found:
168             print(f"      Keine Connection mit matching invitation_msg_id
169                  gefunden")
170             print(f"      Gesuchte invitation_msg_id: {invitation_msg_id}")
171         else:
172             print(f"      Konnte Verifier Connections nicht abrufen")
173         else:
174             print("Holder konnte Invitation nicht akzeptieren")
175         else:
176             print("Fehler beim Erstellen der Invitation")
177
178     # =====
179     # 3. POST-VALIDATION: Connection-Status verifizieren
180     # =====
181     print("\nPost-Validation: Verifizierte Connection-Status...\n")
182
183 validation_success = False
184 verifier_state = None
185 holder_state = None
186
187 if conn_id_verifier_holder:
188     # Prüfe Verifier-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
189     verifier_conn_detail = api_get(
190         VERIFIER_ADMIN_URL,
191         f"/connections/{conn_id_verifier_holder}"
192     )
193
194     if verifier_conn_detail:
195         verifier_state = verifier_conn_detail.get("state")
196         verifier_their_label = verifier_conn_detail.get("their_label")
197         verifier_their_role = verifier_conn_detail.get("their_role")
198         verifier_invitation_msg_id = verifier_conn_detail.get("invitation_msg_id")
199
200         print(f"      Verifier Connection {conn_id_verifier_holder}:")
201         print(f"          State: {verifier_state}")
202         print(f"          Their Label: {verifier_their_label}")

```

```

202     print(f"      Their Role: {verifier_their_role}")
203     print(f"      Invitation Msg ID: {verifier_invitation_msg_id}")
204
205 # Prüfe Holder-Seite (falls conn_id_holder_verifier bekannt)
206 if conn_id_holder_verifier:
207     # Prüfe Holder-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
208     holder_conn_detail = api_get(
209         HOLDER_ADMIN_URL,
210         f"/connections/{conn_id_holder_verifier}"
211     )
212
213 if holder_conn_detail:
214     holder_state = holder_conn_detail.get("state")
215     holder_their_label = holder_conn_detail.get("their_label")
216     holder_their_role = holder_conn_detail.get("their_role")
217     holder_invitation_msg_id = holder_conn_detail.get("invitation_msg_id")
218
219     print(f"\n  Holder Connection {conn_id_holder_verifier}:")
220     print(f"      State: {holder_state}")
221     print(f"      Their Label: {holder_their_label}")
222     print(f"      Their Role: {holder_their_role}")
223     print(f"      Invitation Msg ID: {holder_invitation_msg_id}")
224
225     # Verifizierte dass invitation_msg_id übereinstimmt
226     if invitation_msg_id and verifier_invitation_msg_id ==
227         holder_invitation_msg_id:
228         print(f"      Invitation Msg IDs stimmen überein!")
229
230 # Validierung
231 if verifier_state == "active" and holder_state == "active":
232     validation_success = True
233 elif verifier_state == "active" and not conn_id_holder_verifier:
234     # Bei existierender Connection nur Verifier-Seite bekannt
235     validation_success = True
236
237 # Finale Ausgabe
238 print(f"\n{'='*60}")
239 if validation_success:
240     print(f"CONNECTION ESTABLISHED (did:peer:4): Holder <--> Verifier")
241     print(f"      Verifier Connection ID: {conn_id_verifier_holder}")
242     if conn_id_holder_verifier:
243         print(f"      Holder Connection ID: {conn_id_holder_verifier}")
244     print(f"      Verifier State: {verifier_state}")
245     if holder_state:
246         print(f"      Holder State: {holder_state}")
247     if invitation_msg_id:
248         print(f"      Invitation Msg ID: {invitation_msg_id}")
249 else:
250     print(f"CONNECTION STATUS UNCLEAR")
251     print(f"      Verifier State: {verifier_state if verifier_state else 'N/A'}")
252     print(f"      Holder State: {holder_state if holder_state else 'N/A'}")
253     if conn_id_verifier_holder:
254         print(f"      Verifier Connection ID: {conn_id_verifier_holder}")
255     if conn_id_holder_verifier:
256         print(f"      Holder Connection ID: {conn_id_holder_verifier}")
257 print(f"\n{'='*60}")

```

Listing A-52*Jupyter Notebook Cell 10 Output*

```

1 Connection: Holder --> Verifier erstellen...
2
3 Pre-Check: Prüfe existierende Connections...
4
5     Verifier: 0 Connection(s) gefunden
6     Holder: 1 Connection(s) gefunden
7
8     Keine vollständige existierende Connection gefunden
9     Erstelle neue Connection mit did:peer:4...
10
11 Invitation-Modus: did:peer:4 (P2P Connection)
12 Verifier: Invitation erstellt
13     OOB ID: da555564-e80d-44e7-92ef-265dbfad8da0
14     State: initial
15     Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
16     Services: 1 service(s)
17     Label: Verifier Agent
18     Goal: Establish connection for credential verification
19
20 Holder: Invitation akzeptiert
21     Connection ID (Holder): 925d0cbb-6f5b-43ac-9f64-199961e9af5b
22
23 Verifier sucht Connection via invitation_msg_id...
24     Prüfe 1 Connection(s)...
25
26 Verifier: Connection gefunden via invitation_msg_id!
27     Connection ID: 19df1913-7858-439c-9089-5a947b671c47
28     State: active
29     Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
30     Their Label: Holder Agent
31     Updated At: 2025-12-08T13:39:34.353708Z
32
33 Post-Validation: Verifizierte Connection-Status...
34
35     Verifier Connection 19df1913-7858-439c-9089-5a947b671c47:
36         State: active
37         Their Label: Holder Agent
38         Their Role: invitee
39         Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
40
41 Holder Connection 925d0cbb-6f5b-43ac-9f64-199961e9af5b:
42     State: active
43     Their Label: Verifier Agent
44     Their Role: inviter
45     Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
46     Invitation Msg IDs stimmen überein!
47
48 =====
49 CONNECTION ESTABLISHED (did:peer:4): Holder <--> Verifier
50     Verifier Connection ID: 19df1913-7858-439c-9089-5a947b671c47
51     Holder Connection ID: 925d0cbb-6f5b-43ac-9f64-199961e9af5b

```

```

52     Verifier State: active
53     Holder State: active
54     Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
55 =====

```

Listing A-53*Jupyter Notebook Cell 11*

```

1 # Cell 11: Connection-Übersicht anzeigen
2
3 print("Connection-Übersicht:\n")
4
5 connections_overview = []
6
7 # Issuer Connections
8 issuer_conns = api_get(ISSUER_ADMIN_URL, "/connections")
9 if issuer_conns and "results" in issuer_conns:
10    for conn in issuer_conns["results"]:
11        connections_overview.append({
12            "Agent": "Issuer",
13            "Partner": conn.get("their_label", "Unknown"),
14            "Connection ID": conn["connection_id"],
15            "State": conn["state"],
16            "Invitation Msg ID": conn.get("invitation_msg_id", "N/A") if conn.get("invitation_msg_id") else "N/A",
17            "Status": "active" if conn["state"] == "active" else "N/A"
18        })
19
20 # Holder Connections
21 holder_conns = api_get(HOLDER_ADMIN_URL, "/connections")
22 if holder_conns and "results" in holder_conns:
23    for conn in holder_conns["results"]:
24        partner = conn.get("their_label", "Unknown")
25        connections_overview.append({
26            "Agent": "Holder",
27            "Partner": partner,
28            "Connection ID": conn["connection_id"],
29            "State": conn["state"],
30            "Invitation Msg ID": conn.get("invitation_msg_id", "N/A") if conn.get("invitation_msg_id") else "N/A",
31            "Status": "active" if conn["state"] == "active" else "N/A"
32        })
33
34 # Verifier Connections
35 verifier_conns = api_get(VERIFIER_ADMIN_URL, "/connections")
36 if verifier_conns and "results" in verifier_conns:
37    for conn in verifier_conns["results"]:
38        connections_overview.append({
39            "Agent": "Verifier",
40            "Partner": conn.get("their_label", "Unknown"),
41            "Connection ID": conn["connection_id"],
42            "State": conn["state"],
43            "Invitation Msg ID": conn.get("invitation_msg_id", "N/A") if conn.get("invitation_msg_id") else "N/A",
44            "Status": "active" if conn["state"] == "active" else "N/A"
45        })

```

```

46
47 # Gruppiere nach invitation_msg_id um zusammengehörige Connections zu finden
48 all_connections = []
49 if issuer_conns and "results" in issuer_conns:
50     for conn in issuer_conns["results"]:
51         all_connections.append(("Issuer", conn))
52 if holder_conns and "results" in holder_conns:
53     for conn in holder_conns["results"]:
54         all_connections.append(("Holder", conn))
55 if verifier_conns and "results" in verifier_conns:
56     for conn in verifier_conns["results"]:
57         all_connections.append(("Verifier", conn))
58
59 # Gruppiere nach invitation_msg_id
60 from collections import defaultdict
61 connection_groups = defaultdict(list)
62 for agent, conn in all_connections:
63     invitation_msg_id = conn.get("invitation_msg_id")
64     if invitation_msg_id:
65         connection_groups[invitation_msg_id].append((agent, conn))
66
67 # Zeige gruppierte Connections
68 for idx, (inv_msg_id, group) in enumerate(connection_groups.items(), 1):
69     print(f"Connection Group {idx}:")
70     print(f"    Invitation Msg ID: {inv_msg_id}")
71     for agent, conn in group:
72         print(f"        {agent}: {conn['connection_id']} (State: {conn['state']})")
73     print()

```

Listing A-54

Jupyter Notebook Cell 11 Output

```

1 Connection-Übersicht:
2
3 Connection Group 1:
4     Invitation Msg ID: c22bf8d7-0784-49d0-be53-563a23aa6094
5     Issuer: c5399b17-85f2-4adf-99a5-5aeeee8647d9 (State: active)
6     Holder: 860434c0-7632-41d9-90ea-addaafa68a33 (State: active)
7
8 Connection Group 2:
9     Invitation Msg ID: 6892dae2-a930-47ed-9f5e-cfc23ce856bc
10    Holder: 925d0cbb-6f5b-43ac-9f64-199961e9af5b (State: active)
11    Verifier: 19df1913-7858-439c-9089-5a947b671c47 (State: active)

```

Listing A-55

Jupyter Notebook Cell 12

```

1 # Cell 12 Wallet DID Übersicht anzeigen
2
3 print("Wallet-Übersicht - Alle DIDs:\n")
4
5 # Alle DIDs von allen Agenten abrufen
6 issuer_dids = api_get(ISSUER_ADMIN_URL, "/wallet/did")
7 holder_dids = api_get(HOLDER_ADMIN_URL, "/wallet/did")
8 verifier_dids = api_get(VERIFIER_ADMIN_URL, "/wallet/did")

```

```

9
10 #Zeige originale Responses
11 print("Originale /wallet/did Responses:\n")
12 if issuer_dids:
13     pretty_print(issuer_dids, "Issuer Wallet DIDs")
14 if holder_dids:
15     pretty_print(holder_dids, "Holder Wallet DIDs")
16 if verifier_dids:
17     pretty_print(verifier_dids, "Verifier Wallet DIDs")

```

Listing A-56*Jupyter Notebook Cell 12 Output*

```

1 Wallet-Übersicht - Alle DIDs:
2
3 Originale /wallet/did Responses:
4
5
6 =====
7   Issuer Wallet DIDs
8 =====
9 {
10   "results": [
11     {
12       "did": "did:indy:9pbXiFBZZGwXKp61HQBz3J",
13       "verkey": "2zoa6G7aMfX8GnUEpDxxunFHE7fZktRiiHk1vgMRH2tm",
14       "posture": "posted",
15       "key_type": "ed25519",
16       "method": "indy",
17       "metadata": {
18         "posted": true,
19         "endpoint": "https://host.docker.internal:8020"
20       }
21     },
22     {
23       "did": "did:peer:4zMp1fQXhQPJtqUh7rFgcDJ4pGkqoB9fq7bJeEiLtGnrtZj",
24       "verkey": "4QNeR3HyZsU...",
25       "sJcEJ1ZntCEgzpzLFNEqjUcFjb26CTetW7oYHLG2WbtnjTjPdCWGjAJCPwv5T",
26       "posture": "wallet_only",
27       "key_type": "ml-dsa-65",
28       "method": "did:peer:4",
29       "metadata": {
30         "pqc_enabled": true,
31         "signature_algorithm": "ml-dsa-65",
32         "key_agreement_algorithm": "ml-kem-768",
33         "kem_key_kid": "did:peer:4zMp1fQXhQPJtqUh7rFgcDJ4pGkqoB9fq7bJeEiLtGnrtZj:
34             z25gYmQoBS9XWQb...ZApXw2BMeAy2#key-1",
35         "kem_verkey": "V2cNxGP72k5zsX...ToX",
36         "plugin": "pqc_didpeer4_fm",
37         "version": "0.1.0"
38       }
39     },
40     {
41       "did": "did:peer:4zMp1fQXhQPJtqUh7rFgcDJ4pGkqoB9fq7bJeEiLtGnrtZj:
42             z25gYmQoBS9XWQbLxd...Ay2",
43       "verkey": "4QNeR3HyZsU...WGjAJCPwv5T",

```

```

41     "posture": "wallet_only",
42     "key_type": "ml-dsa-65",
43     "method": "did:peer:4",
44     "metadata": {
45       "pqc_enabled": true,
46       "signature_algorithm": "ml-dsa-65",
47       "key_agreement_algorithm": "ml-kem-768",
48       "kem_key_kid": "did:peer:4zQmP1fQXhQPJtqUh7rFgcDJ4pGkqoB9fq7bJeEiLgNrtZj:
49         z25gYmQoBS9XWQ...ApXw2BMeAy2#key-1",
50       "kem_verkey": "V2cNxGP72k5zsXR1b...Ls7ToX",
51       "plugin": "pqc_didpeer4_fm",
52       "version": "0.1.0"
53     }
54   },
55   {
56     "did": "did:peer:4zQmQUH2nBvShqTckzHpZ4AQPECCvGYuvbY3YY2UwVzizw92:z25gYmQoBS9XW
57       ...GasR6p19J7Yx",
58     "verkey": "JQFmycxRr3Qrqiz8dpbo...yZLiB",
59     "posture": "wallet_only",
60     "key_type": "ml-dsa-65",
61     "method": "did:peer:4",
62     "metadata": {
63       "invitation_reuse": "true",
64       "pqc_enabled": true,
65       "signature_algorithm": "ml-dsa-65",
66       "key_agreement_algorithm": "ml-kem-768",
67       "kem_key_kid": "did:peer:4zQmQUH2nBvShqTckzHpZ4AQPECCvGYuvbY3YY2UwVzizw92:
68         z25gYmQoBS9XWQbLxdKX...R6p19J7Yx#key-1",
69       "kem_verkey": "L9v7qzvr41...ekX45M",
70       "plugin": "pqc_didpeer4_fm",
71       "version": "0.1.0"
72     }
73   ]
74 =====
75   Holder Wallet DIDs
76 =====
77 {
78   "results": [
79     {
80       "did": "did:peer:4zQmaDykrygY16zsPyPzjNfn6TH37xUrH5PmbG6zsbcU5Yd2",
81       "verkey": "4uJptUWFo15yXNCVddmfFJ7...YiQ68EggAH",
82       "posture": "wallet_only",
83       "key_type": "ml-dsa-65",
84       "method": "did:peer:4",
85       "metadata": {
86         "pqc_enabled": true,
87         "signature_algorithm": "ml-dsa-65",
88         "key_agreement_algorithm": "ml-kem-768",
89         "kem_key_kid": "did:peer:4zQmaDykrygY16zsPyPzjNfn6TH37xUrH5PmbG6zsbcU5Yd2:
90           z25gYmQoBS9XWQbLxdK...uBZFKwd5v#key-1",
91         "kem_verkey": "dxUggCJku6PcsEzG4mJ...H17FhgbruM",
92         "plugin": "pqc_didpeer4_fm",
93         "version": "0.1.0"
94       }
95     }
96   ]
97 }
```

```

94     },
95     {
96       "did": "did:peer:4zQmaDykrygYl6ZsPyPzjNFn6TH37xUrH5PmbG6ZsbcU5Yd2:
97         z25gYmQoBS9XWQbLxdK...KuBZFKwd5v",
98       "verkey": "4uJptUWFo15yX...iQ68EggAH",
99       "posture": "wallet_only",
100      "key_type": "ml-dsa-65",
101      "method": "did:peer:4",
102      "metadata": {
103        "pqc_enabled": true,
104        "signature_algorithm": "ml-dsa-65",
105        "key_agreement_algorithm": "ml-kem-768",
106        "kem_key_kid": "did:peer:4zQmaDykrygYl6ZsPyPzjNFn6TH37xUrH5PmbG6ZsbcU5Yd2:
107          z25gYmQoBS9XWQ...FKwd5v#key-1",
108        "kem_verkey": "dxUgqCJk...FhgbruM",
109        "plugin": "pqc_didpeer4_fm",
110        "version": "0.1.0"
111      }
112    },
113    {
114      "did": "did:peer:4zQmdb3wygVriVPEkZFUYfss4JB3Yf2ptp9Y9DoWj45ar5D4",
115      "verkey": "FvTibokwn...YratQBsbPi",
116      "posture": "wallet_only",
117      "key_type": "ml-dsa-65",
118      "method": "did:peer:4",
119      "metadata": {
120        "pqc_enabled": true,
121        "signature_algorithm": "ml-dsa-65",
122        "key_agreement_algorithm": "ml-kem-768",
123        "kem_key_kid": "did:peer:4zQmdb3wygVriVPEkZFUYfss4JB3Yf2ptp9Y9DoWj45ar5D4:
124          z25gYmQoBS9XWQbL...ARuBPZmfJ#key-1",
125        "kem_verkey": "EMjPS1V2X...DAYjGpyEYD",
126        "plugin": "pqc_didpeer4_fm",
127        "version": "0.1.0"
128      }
129    },
130    {
131      "did": "did:peer:4zQmdb3wygVriVPEkZFUYfss4JB3Yf2ptp9Y9DoWj45ar5D4:
132        z25gYmQoBS9XWQb...RuBPZmfJ",
133      "verkey": "FvTibo...DYratQBsbPi",
134      "posture": "wallet_only",
135      "key_type": "ml-dsa-65",
136      "method": "did:peer:4",
137      "metadata": {
138        "pqc_enabled": true,
139        "signature_algorithm": "ml-dsa-65",
140        "key_agreement_algorithm": "ml-kem-768",
141        "kem_key_kid": "did:peer:4zQmdb3wygVriVPEkZFUYfss4JB3Yf2ptp9Y9DoWj45ar5D4:
142          z25gYmQoBS9XWQbL...ybARuBPZmfJ#key-1",
143        "kem_verkey": "EMjPS1...yEYD",
144        "plugin": "pqc_didpeer4_fm",
145        "version": "0.1.0"
146      }
147    }
148  ]
149}

```

```

146 =====
147   Verifier Wallet DIDs
148 =====
149 {
150   "results": [
151     {
152       "did": "did:peer:4zQmP9sGhMY7ziFdBZg2nGHQ4CxtkJ5HGwNNbYuT7QMm4PFV",
153       "verkey": "NgRnsKJrZs...PoWxAqX1z1kM",
154       "posture": "wallet_only",
155       "key_type": "ml-dsa-65",
156       "method": "did:peer:4",
157       "metadata": {
158         "pqc_enabled": true,
159         "signature_algorithm": "ml-dsa-65",
160         "key_agreement_algorithm": "ml-kem-768",
161         "kem_key_kid": "did:peer:4zQmP9sGhMY7ziFdBZg2nGHQ4CxtkJ5HGwNNbYuT7QMm4PFV:
162           z25gYmQoBS9XWQ...B4pXQ3CY#key-1",
163         "kem_verkey": "GJGw4G8...hLKbdh",
164         "plugin": "pqc_didpeer4_fm",
165         "version": "0.1.0"
166       },
167     },
168     {
169       "did": "did:peer:4zQmP9sGhMY7ziFdBZg2nGHQ4CxtkJ5HGwNNbYuT7QMm4PFV:z25gYmQoBS9XW
170       ...Q3CY",
171       "verkey": "NgRn...1z1kM",
172       "posture": "wallet_only",
173       "key_type": "ml-dsa-65",
174       "method": "did:peer:4",
175       "metadata": {
176         "pqc_enabled": true,
177         "signature_algorithm": "ml-dsa-65",
178         "key_agreement_algorithm": "ml-kem-768",
179         "kem_key_kid": "did:peer:4zQmP9sGhMY7ziFdBZg2nGHQ4CxtkJ5HGwNNbYuT7QMm4PFV:
180           z25gYmQ...pXQ3CY#key-1",
181         "kem_verkey": "GJGw4...bdh",
182         "plugin": "pqc_didpeer4_fm",
183         "version": "0.1.0"
184       },
185     },
186     {
187       "did": "did:peer:4zQmRPZ9j5LyppRgJhrow3vfCRZ9CsxuLAJMza2hhhVrbDRG:
188           z25gYmQoBS9XWQbL...EMa",
189       "verkey": "27cheNeC...7AVQs",
190       "posture": "wallet_only",
191       "key_type": "ml-dsa-65",
192       "method": "did:peer:4",
193       "metadata": {
194         "invitation_reuse": "true",
195         "pqc_enabled": true,
196         "signature_algorithm": "ml-dsa-65",
197         "key_agreement_algorithm": "ml-kem-768",
198         "kem_key_kid": "did:peer:4zQmRPZ9j5LyppRgJhrow3vfCRZ9CsxuLAJMza2hhhVrbDRG:
199           z25gYmQoBS9XWQ...shmhEMa#key-1",
200         "kem_verkey": "cNPQCfa2K9...BDf",
201         "plugin": "pqc_didpeer4_fm",
202         "version": "0.1.0"
203       }
204     }
205   ]
206 }
```

```

198         }
199     ]
200 }
201 }
```

Anhang 5.1.5: Teil 5: Credential Issuance - Notfall-Wartungszertifikat

Listing A-57

Jupyter Notebook Cell 13

```

1 # Cell 13: Credential Offer senden (Issuer --> Holder)
2
3 print("Credential Offer senden (Issuer --> Holder)...\\n")
4
5 start_time = time.time()
6
7 # Credential-Daten vorbereiten
8 credential_offer_data = {
9     "comment": "KRITIS Notfall-Wartungszertifikat",
10    "connection_id": conn_id_issuer_holder,
11    "filter": {
12        "indy": {
13            "cred_def_id": cred_def_id
14        }
15    },
16    "credential_preview": {
17        "@type": "issue-credential/2.0/credential-preview",
18        "attributes": [
19            {"name": "first_name", "value": "Max"}, 
20            {"name": "name", "value": "Mustermann"}, 
21            {"name": "organisation", "value": "Musterfirma GmbH"}, 
22            {"name": "role", "value": "Notfalltechniker"}, 
23            {"name": "cert_type", "value": "Notfall-Wartungsberechtigung"}, 
24            {"name": "facility_type", "value": "Umspannwerk Nord-Ost"}, 
25            {"name": "epoch_valid_from", "value": "1765026000"}, 
26            {"name": "epoch_valid_until", "value": "1765033200"}, 
27            {"name": "security_clearance_level", "value": "2"} # Ü2 - Erweiterte
28                Sicherheitsüberprüfung
29        ]
30    },
31    "trace": False
32 }
33
34 # Credential Offer senden
35 cred_offer_response = api_post(
36     ISSUER_ADMIN_URL,
37     "/issue-credential-2.0/send-offer",
38     credential_offer_data
39 )
40
41 if cred_offer_response is not None:
42     cred_ex_id = cred_offer_response.get("cred_ex_id")
43     cred_ex_state = cred_offer_response.get("state")
```

```

43
44     print(f"Credential Offer gesendet")
45     print(f"    Exchange ID: {cred_ex_id}")
46     print(f"    State:       {cred_ex_state}")

47
48     # =====
49     # WARTE AUF CREDENTIAL-AUSSTELLUNG
50     # =====
51     print("\n    Warte auf Credential-Ausstellung (auto-store auf Holder)...")
52     time.sleep(5)

53
54     # =====
55     # STATUS-CHECK AUF BEIDEN SEITEN
56     # =====
57     print("\nStatus-Check:\n")

58
59     # 1. Issuer-Seite Status (BENÖTIGT für Revocation IDs!)
60     cred_status_issuer = api_get(
61         ISSUER_ADMIN_URL,
62         f"/issue-credential-2.0/records/{cred_ex_id}"
63     )

64
65     preserve_flag_active = False
66     issuer_state = None

67
68     if cred_status_issuer:
69         preserve_flag_active = True
70         issuer_state = cred_status_issuer.get("cred_ex_record", {}).get("state")
71         print(f"    Issuer Exchange:")
72         print(f"        State: {issuer_state}")
73         print(f"        --preserve-exchange-records aktiv!")
74     else:
75         print(f"    Issuer Exchange nicht abrufbar")
76         print(f"        --preserve-exchange-records ist NICHT aktiv!")

77
78     # 2. Holder Credentials prüfen
79     print("\n    Holder Credentials:")
80     holder_creds = api_get(HOLDER_ADMIN_URL, "/credentials")

81
82     credential_stored = False
83     credential_referent = None

84
85     if holder_creds and "results" in holder_creds:
86         for cred in holder_creds["results"]:
87             if cred.get("cred_def_id") == cred_def_id:
88                 credential_stored = True
89                 credential_referent = cred.get("referent")
90                 print(f"        Credential ID: {credential_referent}")
91                 print(f"        Schema ID: {cred.get('schema_id', 'N/A')[:50]}...")

92
93             # Zeige Attribute
94             attrs = cred.get("attrs", {})
95             if attrs:
96                 print(f"            Attributes:")
97                 for key, value in list(attrs.items()):
98                     print(f"                - {key}: {value}")
99             break

```

```

100
101 # =====
102 # REVOCATION IDs SPEICHERN (KRITISCH!)
103 # =====
104 rev_reg_id = None
105 cred_rev_id = None
106
107 if cred_status_issuer:
108     indy_data = cred_status_issuer.get("indy", {})
109     rev_reg_id = indy_data.get("rev_reg_id")
110     cred_rev_id = indy_data.get("cred_rev_id")
111
112     if rev_reg_id and cred_rev_id:
113         print("\n" + "="*60)
114         print("REVOCATION-INFORMATIONEN")
115         print("="*60)
116         print(f" Revocation Registry ID: {rev_reg_id}")
117         print(f" Credential Revocation ID: {cred_rev_id}")
118         print(" IDs gespeichert für Revocation-Workflow (Cell 19-21)")
119         print("="*60 + "\n")
120
121 # =====
122 # ISSUER CREDENTIAL REGISTRY
123 # =====
124 print("="*60)
125 print("ISSUER: Alle ausgestellten Credentials")
126 print("="*60)
127
128 all_issuer_records = api_get(ISSUER_ADMIN_URL, "/issue-credential-2.0/records")
129
130 if all_issuer_records and "results" in all_issuer_records:
131     results = all_issuer_records["results"]
132
133 if len(results) > 0:
134     print(f"\nIssuer hat {len(results)} Credential Exchange Record(s)\n")
135
136     for idx, record in enumerate(results, 1):
137         print(f"    Credential #{idx}:")
138         print(f"        Exchange ID: {record['cred_ex_record']['cred_ex_id']}")
139         print(f"        State: {record['cred_ex_record']['state']}")
140
141         # Cred Def ID
142         cred_def_from_record = None
143         if "cred_offer" in record:
144             cred_def_from_record = record.get("cred_offer", {}).get(
145                 "cred_def_id")
146
147         if cred_def_from_record:
148             print(f"        Cred Def ID: {cred_def_from_record[:50]}...")
149
150         # Attributes aus Preview
151         if "cred_preview" in record:
152             attrs = record.get("cred_preview", {}).get("attributes", [])
153             if attrs:
154                 print(f"        Attributes:")
155                 for attr in attrs[:3]:

```

```

155             print(f"      - {attr.get('name')}: {attr.get('value')}\n")
156         if len(attrs) > 3:
157             print(f"      ... und {len(attrs)-3} weitere")
158         print()
159     else:
160         print("\nRegistry ist LEER\n")
161
162     if not preserve_flag_active:
163         print("      --preserve-exchange-records ist NICHT aktiv!")
164         print("      Lösung: docker-compose restart issuer")
165     else:
166         print("      Credential erscheint nach erneutem Ausführen dieser Cell")
167     print()
168
169 # =====
170 # FINALE SUMMARY
171 # =====
172
173 print('='*60)
174 if credential_stored:
175     print(f"KRITIS-NOTFALL-WARTUNGSZERTIFIKAT ERFOLGREICH AUSGESTELLT")
176     print(f"      Issuer State: {issuer_state if issuer_state else 'done'}")
177     print(f"      Credential im Holder Wallet: Yes")
178     print(f"      Credential Referent: {credential_referent}")
179
180     if rev_reg_id and cred_rev_id:
181         print(f"      Revocation IDs gespeichert: Yes")
182     else:
183         print(f"      Revocation IDs nicht gefunden!")
184
185     if preserve_flag_active:
186         print(f"      Issuer Registry verfügbar: Yes")
187     else:
188         print(f"      Issuer Registry verfügbar: NO (restart issuer)")
189     else:
190         print(f"CREDENTIAL STATUS UNKLAR")
191         print(f"      Issuer State: {issuer_state if issuer_state else 'N/A'}")
192         print(f"      Credential im Wallet: NO")
193
194 print(f'='*60)
195
196 # Zeige vollständige Response
197 # pretty_print(cred_status_issuer, "Credential Offer Response (KRITIS)")
198
199 else:
200     print("Fehler beim Senden des Credential Offers")

```

Listing A-58

Jupyter Notebook Cell 13 Output

```

1 Credential Offer senden (Issuer --> Holder)...
2
3 Credential Offer gesendet
4   Exchange ID: 4cd96e9e-ec5f-4b54-8df3-2ee1d5e0607d
5   State:       offer-sent

```

```

6
7 Warte auf Credential-Ausstellung (auto-store auf Holder)...
8
9 Status-Check:
10
11 Issuer Exchange:
12     State: done
13     --preserve-exchange-records aktiv!
14
15 Holder Credentials:
16     Credential ID: 39ac5fc4-efc2-45eb-9a21-01c589757b65
17     Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintena...
18     Attributes:
19         - facility_type: Umspannwerk Nord-Ost
20         - epoch_valid_from: 1765026000
21         - organisation: Musterfirma GmbH
22         - cert_type: Notfall-Wartungsberechtigung
23         - first_name: Max
24         - role: Notfalltechniker
25         - epoch_valid_until: 1765033200
26         - security_clearance_level: 2
27         - name: Mustermann
28
29 =====
30 REVOCATION-INFORMATIONEN
31 =====
32     Revocation Registry ID: 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:
33         default:CL_ACCUM:9efc6971-b713-465f-85db-a192b107b73e
34     Credential Revocation ID: 1
35     IDs gespeichert für Revocation-Workflow (Cell 19-21)
36
37 =====
38 ISSUER: Alle ausgestellten Credentials
39 =====
40
41 Issuer hat 1 Credential Exchange Record(s)
42
43     Credential #1:
44         Exchange ID: 4cd96e9e-ec5f-4b54-8df3-2ee1d5e0607d
45         State: done
46
47 =====
48 KRITIS-NOTFALL-WARTUNGSZERTIFIKAT ERFOLGREICH AUSGESTELLT
49     Issuer State: done
50     Credential im Holder Wallet: Yes
51     Credential Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
52     Revocation IDs gespeichert: Yes
53     Issuer Registry verfügbar: Yes
54 =====

```

Listing A-59

Jupyter Notebook Cell 14

```

1 # Cell 14: Holder Credentials im Detail anzeigen
2

```

```

3 print("Holder Wallet - Gespeicherte Credentials:\n")
4
5 holder_creds = api_get(HOLDER_ADMIN_URL, "/credentials")
6
7 if holder_creds and "results" in holder_creds and len(holder_creds["results"]) > 0:
8     for idx, cred in enumerate(holder_creds["results"], 1):
9         print(f"\n{'='*60}")
10        print(f"Credential #{idx}")
11        print(f"{'='*60}")
12        print(f"Referent: {cred.get('referent', 'N/A')}")
13        print(f"Schema ID: {cred.get('schema_id', 'N/A')}")
14        print(f"Cred Def ID: {cred.get('cred_def_id', 'N/A')}")
15
16        # Credentials Revoked Abfrage gegen /credential/revoked/{credential_id ==>
17        # Referent}
18        print(f"Revoked Status: {api_get(HOLDER_ADMIN_URL, f'/credential/revoked/{cred
19            .get('referent', 'N/A')}').get('revoked', 'N/A')}")
20
21    if "attrs" in cred:
22        print("\nAttribute:")
23        for attr_name, attr_value in cred["attrs"].items():
24            print(f" - {attr_name}: {attr_value}")
25 else:
26     print("Holder hat noch keine Credentials")

```

Listing A-60

Jupyter Notebook Cell 14 Output

```

1 Holder Wallet - Gespeicherte Credentials:
2
3 =====
4 =====
5 Credential #1
6 =====
7 Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
8 Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
9 Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
10 Revoked Status: False
11
12 Attribute:
13   - first_name: Max
14   - cert_type: Notfall-Wartungsberechtigung
15   - organisation: Musterfirma GmbH
16   - epoch_valid_from: 1765026000
17   - epoch_valid_until: 1765033200
18   - role: Notfalltechniker
19   - facility_type: Umspannwerk Nord-Ost
20   - security_clearance_level: 2
21   - name: Mustermann

```

Anhang 5.1.6: Teil 6: Proof Presentation - Zutrittskontrolle am Umspannwerk

Listing A-61

Jupyter Notebook Cell 15

```

1 # Cell 15: Proof Request senden (Verifier --> Holder) - mit REVOCATION ALWAYS-ON
2
3 print("Proof Request senden (Verifier --> Holder)...\\n")
4
5 start_time = time.time()
6
7 # Get current timestamp for non-revocation interval
8 current_timestamp = int(time.time())
9
10 # Proof Request Data (ACA-Py API) - Privacy-Preserving ZKP!
11 # API: POST /present-proof-2.0/send-request
12 # REVEALED: cert_type, facility_type, epoch_valid_from, epoch_valid_until, role (5
# Attribute)
13 # ZKP PREDICATE: security_clearance_level >= 2 (Ü2) - OHNE Offenlegung der exakten
# Stufe!
14 # UNREVEALED: first_name, name, organisation (3 Attribute - ZKP ohne Offenlegung!)
15 proof_request_data = {
16     "connection_id": conn_id_verifier_holder, # from Cell 15
17     "presentation_request": {
18         "indy": {
19             "name": "Proof of KRITIS Emergency Maintenance Authorization",
20             "version": "1.0",
21             "requested_attributes": {
22                 "attr1_referent": {
23                     "name": "cert_type",
24                     "restrictions": [{"cred_def_id": cred_def_id}],
25                     "non_revoked": {"from": 0, "to": current_timestamp}
26                 },
27                 "attr2_referent": {
28                     "name": "facility_type",
29                     "restrictions": [{"cred_def_id": cred_def_id}],
30                     "non_revoked": {"from": 0, "to": current_timestamp}
31                 },
32                 "attr3_referent": {
33                     "name": "epoch_valid_from",
34                     "restrictions": [{"cred_def_id": cred_def_id}],
35                     "non_revoked": {"from": 0, "to": current_timestamp}
36                 },
37                 "attr4_referent": {
38                     "name": "epoch_valid_until",
39                     "restrictions": [{"cred_def_id": cred_def_id}],
40                     "non_revoked": {"from": 0, "to": current_timestamp}
41                 },
42                 "attr5_referent": {
43                     "name": "role",
44                     "restrictions": [{"cred_def_id": cred_def_id}],
45                     "non_revoked": {"from": 0, "to": current_timestamp}
46                 }
47             },
48             "requested_predicates": {
49                 "pred1_clearance": {

```

```

50         "name": "security_clearance_level",
51         "p_type": ">=",
52         "p_value": 2,
53         "restrictions": [{"cred_def_id": cred_def_id}],
54         "non_revoked": {"from": 0, "to": current_timestamp}
55     }
56   }
57 }
58 }
59 }
60
61 print(f"Proof Request für KRITIS-Zutrittsberechtigung:")
62 print(f"  Endpoint:      /present-proof-2.0/send-request")
63 print(f"  Connection:    {conn_id_verifier_holder}")
64 print(f"  Cred Def ID:  {cred_def_id}")
65 print(f"  Proof Name:   Proof of KRITIS Emergency Maintenance Authorization")
66 print(f"  REVEALED:      Berechtigung, Anlage, Sicherheitsstufe, Gültigkeitszeitraum,
67       Rolle (6 Attribute)")
68 print(f"  DATENSCHUTZ: Identität (Vor-/Nachname) und Organisation werden NICHT
69       offengelegt (Selective-Disclosure)\n")
70
71 # Send Proof Request (ACA-Py endpoint)
72 proof_request_response = api_post(
73     VERIFIER_ADMIN_URL,
74     "/present-proof-2.0/send-request",
75     proof_request_data
76 )
77
78 if proof_request_response is not None:
79     # Response Format: {"pres_ex_id": "...", "state": "...", ...}
80     pres_ex_id = proof_request_response.get("pres_ex_id")
81     pres_ex_state = proof_request_response.get("state")
82
83     print(f"Proof Request gesendet")
84     print(f"  Exchange ID: {pres_ex_id}")
85     print(f"  State:        {pres_ex_state}")
86
87     # Show full Proof Request Response
88     # pretty_print(proof_request_response, "Proof Request Response (KRITIS)")
89 else:
90     print("Fehler beim Senden des Proof Requests")

```

Listing A-62

Jupyter Notebook Cell 15 Output

```

1 Proof Request senden (Verifier --> Holder)...
2
3 Proof Request für KRITIS-Zutrittsberechtigung:
4   Endpoint:      /present-proof-2.0/send-request
5   Connection:    19df1913-7858-439c-9089-5a947b671c47
6   Cred Def ID:  9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
7   Proof Name:   Proof of KRITIS Emergency Maintenance Authorization
8   REVEALED:      Berechtigung, Anlage, Sicherheitsstufe, Gültigkeitszeitraum, Rolle
                  (6 Attribute)

```

```

9  DATENSCHUTZ: Identität (Vor-/Nachname) und Organisation werden NICHT offen gelegt ( 
10 Selective-Disclosure)
11 Proof Request gesendet
12   Exchange ID: 64b89588-48f2-4316-b363-d1bc6afcc841
13   State:       request-sent

```

Listing A-63

Jupyter Notebook Cell 16

```

1 # Cell 16: Holder - Proof Request empfangen und Credentials auswählen
2
3 print("Holder: Proof Request verarbeiten...\n")
4
5 if pres_ex_id:
6     # Holder prüft empfangene Proof Requests
7     time.sleep(2)
8
9 holder_pres_ex = api_get(HOLDER_ADMIN_URL, "/present-proof-2.0/records")
10
11 if holder_pres_ex and "results" in holder_pres_ex:
12     # Finde den aktuellen Presentation Exchange
13     current_pres = None
14     for pres in holder_pres_ex["results"]:
15         if pres.get("state") == "request-received":
16             current_pres = pres
17             holder_pres_ex_id = pres["pres_ex_id"]
18             break
19
20     if current_pres:
21         print(f" Holder hat Proof Request empfangen")
22         print(f"   Presentation Exchange ID: {holder_pres_ex_id}")
23
24     # Credentials für die Presentation abrufen
25     creds_for_pres = api_get(
26         HOLDER_ADMIN_URL,
27         f"/present-proof-2.0/records/{holder_pres_ex_id}/credentials"
28     )
29
30     if creds_for_pres:
31         print(f"\n Verfügbare Credentials für Presentation:")
32
33     # API kann entweder list oder dict zurückgeben
34     total_creds = 0
35
36     if isinstance(creds_for_pres, list):
37         # Liste von Credential Records
38         total_creds = len(creds_for_pres)
39         print(f"   Gefunden: {total_creds} Credential(s)")
40
41     # Zeige Details der ersten paar Credentials
42     for idx, cred in enumerate(creds_for_pres[:3], 1):
43         print(f"\n   Credential #{idx}:")
44         if "cred_info" in cred:
45             cred_info = cred["cred_info"]

```

```

46             print(f"      Referent: {cred_info.get('referent', 'N/A')}")
47             print(f"      Schema ID: {cred_info.get('schema_id', 'N/A')}")
48             print(f"      Cred Def ID: {cred_info.get('cred_def_id', 'N/A')} )")
49
50     # Zeige Attribute
51     attrs = cred_info.get("attrs", {})
52     if attrs:
53         print(f"      Attributes:")
54         # Zeige erste 3 Attribute
55         shown = 0
56         for key, value in attrs.items():
57             print(f"          - {key}: {value}")
58
59     elif isinstance(creds_for_pres, dict):
60         # Dictionary mit referent als key
61         for referent, creds in creds_for_pres.items():
62             if isinstance(creds, list):
63                 total_creds += len(creds)
64                 print(f"      {referent}: {len(creds)} Credential(s)")
65
66         # Zeige Details des ersten Credentials
67         if len(creds) > 0:
68             cred = creds[0]
69             if "cred_info" in cred:
70                 cred_info = cred["cred_info"]
71                 print(f"      Cred Def ID: {cred_info.get('"
72                     cred_def_id', 'N/A')} )")
73                 attrs = cred_info.get("attrs", {})
74                 if attrs:
75                     attr_count = len(attrs)
76                     print(f"      {attr_count} Attribute(s) verfü"
77                         gbar")
78             else:
79                 print(f"      Unbekanntes Format: {type(creds_for_pres).__name__}")
80             total_creds = 0
81
82         print(f"\n Total: {total_creds} passende Credential(s) gefunden")
83
84         if total_creds == 0:
85             print(f"\n WARNUNG: Keine passenden Credentials gefunden!")
86             print(f"      Stelle sicher dass Cell 16 erfolgreich war")
87             print(f"      Prüfe ob Credential im Holder Wallet ist (Cell 16
88                   Output)")
89             print(f"      Schema/Cred Def müssen mit Proof Request ü
90                   bereinstimmen")
91         else:
92             print(f"\n  Keine Credentials für Presentation verfügbar")
93             print(f"  API-Antwort war leer oder None")
94     else:
95         print("  Kein Proof Request im Status 'request-received'")
96         print("\n  Verfügbare Presentation Exchanges:")
97         for pres in holder_pres_ex["results"]:
98             print(f"      {pres['pres_ex_id']} State: {pres.get('state')}")
99     else:

```

```

96     print("  Keine Presentation Exchanges gefunden")
97     print("  Führe Cell 18 aus um Proof Request zu senden")
98 else:
99     print("  Übersprungen: Kein Presentation Exchange vorhanden")
100    print("  Führe Cell 18 aus um Proof Request zu erstellen")

```

Listing A-64*Jupyter Notebook Cell 16 Output*

```

1 Holder: Proof Request verarbeiten...
2
3 Holder hat Proof Request empfangen
4   Presentation Exchange ID: fb85c70d-79c9-40a7-b0cd-d75c032cf86
5
6 Verfügbare Credentials für Presentation:
7   Gefunden: 1 Credential(s)
8
9 Credential #1:
10  Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
11  Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
12  Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
13  Attributes:
14    - facility_type: Umspannwerk Nord-Ost
15    - organisation: Musterfirma GmbH
16    - epoch_valid_until: 1765033200
17    - name: Mustermann
18    - epoch_valid_from: 1765026000
19    - cert_type: Notfall-Wartungsberechtigung
20    - role: Notfalltechniker
21    - security_clearance_level: 2
22    - first_name: Max
23
24 Total: 1 passende Credential(s) gefunden

```

Listing A-65*Jupyter Notebook Cell 17*

```

1 # Cell 17: Holder - Presentation senden
2
3 print("Holder: Presentation senden...\n")
4
5 if pres_ex_id and holder_pres_ex_id:
6     start_time = time.time()
7
8     # Hole Credentials nochmal
9     creds_for_pres = api_get(
10         HOLDER_ADMIN_URL,
11         f"/present-proof-2.0/records/{holder_pres_ex_id}/credentials"
12     )
13
14 if creds_for_pres:
15     print(f"Credentials für Presentation gefunden")
16
17     # Baue Presentation Request
18     requested_credentials = {

```



```

69         requested_credentials["indy"]["requested_predicates"][referent
70             ] = {
71                 "cred_id": cred_id
72             }
73             print(f"    Mapped predicate '{referent}' --> Credential {
74                 cred_id[:20]}...")
75
76     # Prüfe ob wir Credentials gemappt haben
77     total_mapped = (
78         len(requested_credentials["indy"]["requested_attributes"]) +
79         len(requested_credentials["indy"]["requested_predicates"]))
80
81     if total_mapped == 0:
82         print("\n  WARNUNG: Keine Credentials gemappt!")
83         print("  Versuche automatisches Mapping...\n")
84
85     # Fallback: Verwende erstes verfügbares Credential für alle Requests
86     # Hole Proof Request Details
87     pres_ex_details = api_get(
88         HOLDER_ADMIN_URL,
89         f"/present-proof-2.0/records/{holder_pres_ex_id}")
90
91     if pres_ex_details and "pres_request" in pres_ex_details:
92         pres_request = pres_ex_details["pres_request"]
93
94     # Hole erstes verfügbares Credential
95     first_cred_id = None
96     if isinstance(creds_for_pres, list) and len(creds_for_pres) > 0:
97         first_cred_id = creds_for_pres[0].get("cred_info", {}).get("referent")
98     elif isinstance(creds_for_pres, dict):
99         for creds in creds_for_pres.values():
100             if isinstance(creds, list) and len(creds) > 0:
101                 first_cred_id = creds[0].get("cred_info", {}).get("referent")
102                 break
103
104     if first_cred_id:
105         # Map alle requested attributes
106         if "requested_attributes" in pres_request:
107             for attr_ref in pres_request["requested_attributes"].keys():
108                 requested_credentials["indy"]["requested_attributes"][
109                     attr_ref] = {
110                         "cred_id": first_cred_id,
111                         "revealed": True
112                     }
113                     print(f"    Auto-mapped attribute '{attr_ref}' --> {
114                         first_cred_id[:20]}...")
115
116         # Map alle requested predicates
117         if "requested_predicates" in pres_request:
118             for pred_ref in pres_request["requested_predicates"].keys():
119                 requested_credentials["indy"]["requested_predicates"][
120                     pred_ref] = {
121                         "cred_id": first_cred_id

```

```

119
120         }
121         print(f"  Auto-mapped predicate '{pred_ref}' --> {
122             first_cred_id[:20]}...")
123
124     total_mapped = (
125         len(requested_credentials["indy"]["requested_attributes"]) +
126         len(requested_credentials["indy"]["requested_predicates"]))
127
128     print(f"\nPresentation Mapping:")
129     print(f"  Requested Attributes: {len(requested_credentials['indy']['
130         requested_attributes'])}")
131     print(f"  Requested Predicates: {len(requested_credentials['indy']['
132         requested_predicates'])}")
133     print(f"  Total Mapped: {total_mapped}")
134
135     if total_mapped > 0:
136         # Sende Presentation
137         print("\nSende Presentation...")
138         presentation_response = api_post(
139             HOLDER_ADMIN_URL,
140             f"/present-proof-2.0/records/{holder_pres_ex_id}/send-presentation",
141             requested_credentials
142         )
143
144         if presentation_response is not None:
145             print(f"\nHolder hat Presentation gesendet")
146             print(f"  Presentation Exchange ID: {holder_pres_ex_id}")
147             print(f"  State: {presentation_response.get('state', 'N/A')}")
148             print(f"  Role: {presentation_response.get('role', 'N/A')}")
149
150             # Warte auf Verification
151             print("\n  Warte auf Verification durch Verifier...")
152             time.sleep(3)
153
154             # Prüfe finalen Status (kann 404 geben wenn abgeschlossen!)
155             final_status = api_get(
156                 HOLDER_ADMIN_URL,
157                 f"/present-proof-2.0/records/{holder_pres_ex_id}"
158             )
159
160             if final_status:
161                 # Record noch vorhanden (--preserve-exchange-records aktiv)
162                 final_state = final_status.get("state")
163                 print(f"\n  Finaler Holder Status: {final_state}")
164
165                 if final_state in ["done", "presentation-sent"]:
166                     print(f"    Presentation erfolgreich abgeschlossen!")
167                     print(f"    Record bleibt verfügbar (--preserve-exchange-
168                         records aktiv)")
169                 elif final_state == "abandoned":
170                     print(f"    Presentation wurde abgebrochen")
171                 else:
172                     print(f"    Presentation in Bearbeitung...")
173             else:
174                 # 404 Error = Record wurde gelöscht = erfolgreich abgeschlossen!
175                 print(f"\n  Finaler Holder Status: completed/archived (404)")

```

```

172         print(f"    Presentation erfolgreich abgeschlossen!")
173         print(f"    Record wurde nach Completion gelöscht (--preserve-
174             exchange-records nicht aktiv)")
175         print(f"\n    Hinweis: Für dauerhafte Records, starte Holder neu:")
176             )
177         print(f"    docker-compose restart holder")

178     else:
179         print("\n Fehler beim Senden der Presentation")
180         print("    API-Response war leer oder fehlerhaft")
181     else:
182         print("\n Kann Presentation nicht senden - keine Credentials gemappt")
183         print("    Prüfe ob Credentials die Proof Request Anforderungen erfüllen")
184         print("    Prüfe Schema und Cred Def IDs")
185     else:
186         print("    Keine Credentials für Presentation gefunden")
187         print("    Führe Cell 16 aus um Credential auszustellen")
188     else:
189         print("    Übersprungen: Presentation Exchange nicht vorhanden")
190     if not pres_ex_id:
191         print("    Verifier Presentation Exchange ID fehlt (führe Cell 18 aus)")
192     if not holder_pres_ex_id:
193         print("    Holder Presentation Exchange ID fehlt (führe Cell 19 aus)")

```

Listing A-66*Jupyter Notebook Cell 17 Output*

```

1 Holder: Presentation senden...
2
3 Credentials für Presentation gefunden
4 Format: Liste mit 1 Credential(s)
5
6 Mapped attribute 'attr3_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
7 Mapped attribute 'attr1_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
8 Mapped predicate 'pred1_clearance' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
9 Mapped attribute 'attr4_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
10 Mapped attribute 'attr2_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
11 Mapped attribute 'attr5_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
12
13 Presentation Mapping:
14     Requested Attributes: 5
15     Requested Predicates: 1
16     Total Mapped: 6
17
18 Sende Presentation...
19
20 Holder hat Presentation gesendet
21     Presentation Exchange ID: fb85c70d-79c9-40a7-b0cd-d75c032cf86
22     State: presentation-sent

```

```

23     Role: prover
24
25     Warte auf Verification durch Verifier...
26
27     Finaler Holder Status: done
28     Presentation erfolgreich abgeschlossen!
29     Record bleibt verfügbar (--preserve-exchange-records aktiv)

```

Listing A-67*Jupyter Notebook Cell 18*

```

1 # Cell 18: Verifier - Presentation verifizieren (mit Revocation Detection + Zeitgü
  ltigkeit)
2
3 print(" Presentation verifizieren (Verifier)...\n")
4
5 start_time = time.time()
6
7 # Wait for presentation to be received
8 import time as t
9 t.sleep(5) # Give Holder time to generate and send proof
10
11 # Get presentation exchange record (ACA-Py endpoint)
12 # API: GET /present-proof-2.0/records/{pres_ex_id}
13 pres_ex_record = api_get(
14     VERIFIER_ADMIN_URL,
15     f"/present-proof-2.0/records/{pres_ex_id}"
16 )
17
18 if pres_ex_record is not None:
19     # Response Format: {"state": "...", "verified": "...", "pres": {...}, "by_format":
20         {...}, ...}
21     pres_state = pres_ex_record.get("state")
22     pres_verified = pres_ex_record.get("verified")
23
24     # =====
25     # ATTRIBUTE MAPPING: referent -> name -> value
26     # =====
27     # Step 1: Get requested_attributes from by_format.pres_request.indy (referent ->
28         # name mapping)
29     by_format = pres_ex_record.get("by_format", {})
30     pres_request_indy = by_format.get("pres_request", {}).get("indy", {})
31     requested_attributes = pres_request_indy.get("requested_attributes", {})
32
33     # Step 2: Get revealedAttrs from by_format.pres.indy.requested_proof (referent ->
34         # value mapping)
35     pres_indy = by_format.get("pres", {}).get("indy", {})
36     requested_proof = pres_indy.get("requested_proof", {})
37     revealedAttrs_by_referent = requested_proof.get("revealedAttrs", {})
38
39     # Step 3: Build name -> value mapping
40     revealedAttrs = {}
41     for referent, attr_data in revealedAttrs_by_referent.items():
42         # Get the attribute name from requested_attributes
43         if referent in requested_attributes:
44             attr_name = requested_attributes[referent].get("name")

```

```

42         attr_value = attr_data.get("raw")
43         if attr_name and attr_value:
44             revealedAttrs[attr_name] = attr_value
45
46     print(f" Proof erfolgreich verifiziert!")
47
48     print(f"\nVerifizierte Attribute (REVEALED):")
49     for name, value in revealedAttrs.items():
50         print(f" - {name}: {value}")
51
52     print(f"\n PER ZKP VERIFIZIERT ABER DURCH DATENSCHUTZ GESCHÜTZT (UNREVEALED):")
53     print(f" - Vorname: NICHT offengelegt (Zero-Knowledge-Proof)")
54     print(f" - Nachname: NICHT offengelegt (Zero-Knowledge-Proof)")
55     print(f" - Organisation: NICHT offengelegt (Zero-Knowledge-Proof)")
56
57     print(f"\n State:      {pres_state}")
58     print(f" Verified:   {pres_verified}")
59
60     # =====
61     # REVOCATION CHECK
62     # =====
63     is_revoked = False
64     if pres_state == "done" and pres_verified == "true":
65         print(f"\n Credential ist NICHT revoked (gültig)")
66     else:
67         print(f"\n Credential ist REVOKED!")
68     is_revoked = True
69
70     # =====
71     # ZEITGÜLTIGKEITS-PRÜFUNG
72     # =====
73     print("\n" + "="*60)
74     print("ZEITGÜLTIGKEITS-PRÜFUNG")
75     print("="*60)
76
77     current_epoch = 1765029600 # Beispiel: Fester Zeitpunkt für Test (2024-09-05
78     12:00:00 UTC)
79     epoch_valid_from = None
80     epoch_valid_until = None
81     is_time_valid = False
82
83     # Extrahiere epoch_valid_from und epoch_valid_until aus revealedAttrs (name ->
84     # value mapping)
85     if "epoch_valid_from" in revealedAttrs:
86         epoch_valid_from = int(revealedAttrs["epoch_valid_from"])
87
88     if "epoch_valid_until" in revealedAttrs:
89         epoch_valid_until = int(revealedAttrs["epoch_valid_until"])
90
91     if epoch_valid_from is not None and epoch_valid_until is not None:
92         # Konvertiere zu lesbaren Timestamps
93         from datetime import datetime
94         valid_from_dt = datetime.fromtimestamp(epoch_valid_from)
95         valid_until_dt = datetime.fromtimestamp(epoch_valid_until)
96         current_dt = datetime.fromtimestamp(current_epoch)

```

```

96     print(f"    Aktueller Zeitpunkt: {current_dt.strftime('%Y-%m-%d %H:%M:%S')} ) ("
97         Epoch: {current_epoch}) ==> Beispielwert")
98     print(f"    Gültig ab:           {valid_from_dt.strftime('%Y-%m-%d %H:%M:%S')} )"
99         (Epoch: {epoch_valid_from}))")
100    print(f"    Gültig bis:          {valid_until_dt.strftime('%Y-%m-%d %H:%M:%S')} )"
101        } (Epoch: {epoch_valid_until}))")

102    # Prüfe Zeitgültigkeit
103    if epoch_valid_from <= current_epoch <= epoch_valid_until:
104        is_time_valid = True
105        print(f"\n        Zertifikat ist ZEITLICH GÜLTIG")
106    else:
107        is_time_valid = False
108        if current_epoch < epoch_valid_from:
109            print(f"\n        Zertifikat ist NOCH NICHT gültig (zu früh)")
110        else:
111            print(f"\n        Zertifikat ist ABGELAUFEN (zu spät)")
112    else:
113        print(f"        Zeitgültigkeits-Attribute nicht gefunden!")
114        print(f"        epoch_valid_from: {'gefunden' if epoch_valid_from else 'FEHLT'}")
115        )
116        print(f"        epoch_valid_until: {'gefunden' if epoch_valid_until else 'FEHLT'}")
117
118    print("=". * 60)

119    # =====
120    # PREDICATE AUSWERTUNG (ZKP)
121    # =====
122    print("\n" + "=". * 60)
123    print("ZERO-KNOWLEDGE-PROOF AUSWERTUNG")
124    print("=". * 60)

125    predicates = requested_proof.get("predicates", {})
126    has_required_clearance = False
127
128    if predicates:
129        for ref, pred_data in predicates.items():
130            print(f"        Predicate erfüllt: security_clearance_level >= 2 (Ü2)")
131            print(f"        Zero-Knowledge-Proof: Exakte Sicherheitsstufe NICHT offengelegt"
132                  !")
133            print(f"        Techniker hat mindestens Ü2 (Erweiterte Sicherheitsüberprüfung)"
134                  ")
135            has_required_clearance = True
136    else:
137        print(f"        Keine Predicates im Proof gefunden!")

138
139    # =====
140    # FINALE ZUGRIFFSENTSCHEIDUNG
141    # =====
142    print("\n" + "=". * 60)
143    print("FINALE ZUGRIFFSENTSCHEIDUNG")
144    print("=". * 60)

```

```

146     # Alle drei Bedingungen müssen erfüllt sein
147     if not is_revoked and is_time_valid and has_required_clearance:
148         print(f"\n ZUGANG GEWÄHRT")
149         print(f"    Credential ist gültig (nicht revoked)")
150         print(f"    Zertifikat ist zeitlich gültig")
151         print(f"    Sicherheitsfreigabe >= Ü2 (Zero-Knowledge-Proof)")
152         print(f"\n    Zugang zum Umspannwerk Nord-Ost GEWÄHRT")
153     else:
154         print(f"\n ZUGANG VERWEIGERT")
155         if is_revoked:
156             print(f"    Credential ist REVOKED")
157         if not is_time_valid:
158             print(f"    Zertifikat ist NICHT zeitlich gültig")
159         if not has_required_clearance:
160             print(f"    Sicherheitsfreigabe NICHT ausreichend (< Ü2)")
161             print(f"\n    Zugang zum Umspannwerk Nord-Ost VERWEIGERT")
162
163     print("="*60)
164
165     print(f"\n Privacy-Preserving Verification erfolgreich (DSGVO-konform)")
166
167     # Show full Presentation Record
168     # pretty_print(pres_ex_record, "Presentation Record (KRITIS)")
169
170 else:
171     print(" Fehler beim Abrufen der Presentation")

```

Listing A-68

Jupyter Notebook Cell 18 Output

```

1 Presentation verifizieren (Verifier)...
2
3 Proof erfolgreich verifiziert!
4
5 Verifizierte Attribute (REVEALED):
6   - facility_type: Umspannwerk Nord-Ost
7   - epoch_valid_until: 1765033200
8   - cert_type: Notfall-Wartungsberechtigung
9   - role: Notfalltechniker
10  - epoch_valid_from: 1765026000
11
12 PER ZKP VERIFIZIERT ABER DURCH DATENSCHUTZ GESCHÜTZT (UNREVEALED):
13   - Vorname: NICHT offengelegt (Zero-Knowledge-Proof)
14   - Nachname: NICHT offengelegt (Zero-Knowledge-Proof)
15   - Organisation: NICHT offengelegt (Zero-Knowledge-Proof)
16
17 State:      done
18 Verified:   true
19
20 Credential ist NICHT revoked (gültig)
21
22 =====
23 ZEITGÜLTIGKEITS-PRÜFUNG
24 =====
25   Aktueller Zeitpunkt: 2025-12-06 14:00:00 (Epoch: 1765029600) ==> Beispielwert
26   Gültig ab:           2025-12-06 13:00:00 (Epoch: 1765026000)

```

```

27 Gültig bis: 2025-12-06 15:00:00 (Epoch: 1765033200)
28
29 Zertifikat ist ZEITLICH GÜLTIG
30 =====
31 =====
32 =====
33 ZERO-KNOWLEDGE-PROOF AUSWERTUNG
34 =====
35 Predicate erfüllt: security_clearance_level >= 2 (Ü2)
36 Zero-Knowledge-Proof: Exakte Sicherheitsstufe NICHT offengelegt!
37 Techniker hat mindestens Ü2 (Erweiterte Sicherheitsüberprüfung)
38 =====
39
40 =====
41 FINALE ZUGRIFFSENTSCHEIDUNG
42 =====
43
44 ZUGANG GEWÄHRT
45 Credential ist gültig (nicht revoked)
46 Zertifikat ist zeitlich gültig
47 Sicherheitsfreigabe >= Ü2 (Zero-Knowledge-Proof)
48
49 Zugang zum Umspannwerk Nord-Ost GEWÄHRT
50 =====
51
52 Privacy-Preserving Verification erfolgreich (DSGVO-konform)

```

Anhang 5.1.7: Teil 7: Revocation & Deletion- Beendigung der Wartungsberechtigung

Listing A-69

Jupyter Notebook Cell 19

```

1 # Cell 19: Revocation Registries anzeigen
2
3 print("=="*60)
4 print("ISSUER: Revocation Registries")
5 print("=="*60 + "\n")
6
7 registries = api_get(
8     ISSUER_ADMIN_URL,
9     "/revocation/registries/created?state=active"
10 )
11
12 if registries and "rev_reg_ids" in registries:
13     reg_ids = registries["rev_reg_ids"]
14     print(f"Aktive Revocation Registries: {len(reg_ids)}\n")
15
16     for idx, reg_id in enumerate(reg_ids, 1):
17         print(f"Registry #{idx}: {reg_id}")
18
19     # Registry Details abrufen

```

```
20     reg_info = api_get(ISSUER_ADMIN_URL, f"/revocation/registry/{reg_id}").get('
21         result', 'N/A')
22
23     if reg_info:
24         print(f"    State: {reg_info.get('state')}")
25         print(f"    Max Credentials: {reg_info.get('max_cred_num')}")
26         print(f"    Issuer DID: {reg_info.get('issuer_did', 'N/A')}")
27         print(f"    Tails Hash: {reg_info.get('tails_hash', 'N/A')}")
28         print(f"    Tails Location: {reg_info.get('tails_local_path', 'N/A')}")
29         print(f"    Created: {reg_info.get('created_at', 'N/A')}\n")
30
31     else:
32
33     print("Keine aktiven Revocation Registries gefunden")
34     print("    Stelle sicher dass Cell 7 mit support_revocation: True läuft")
35     print("    Führe Cell 13 aus um Credential mit Revocation auszustellen")
36
37     print("="*60)
```

Listing A-70

Jupyter Notebook Cell 19 Output

```
1 =====
2 ISSUER: Revocation Registries
3 =====
4
5 Aktive Revocation Registries: 2
6
7 Registry #1: 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:9
8   efc6971-b713-465f-85db-a192b107b73e
9   State: active
10  Max Credentials: 100
11  Issuer DID: did:indy:9pbXiFBZZGwXKp61HQBz3J
12  Tails Hash: 34NrwsR9LbHRHoarwVMmTP95Zr7KJhzqqFzCpFRUw1K4
13  Tails Location: /home/aries/.indy_client/tails/9pbXiFBZZGwXKp61HQBz3J:4:9
14    pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:9efc6971-b713-465f-85db-
15      a192b107b73e/34NrwsR9LbHRHoarwVMmTP95Zr7KJhzqqFzCpFRUw1K4
16  Created: 2025-12-08T13:03:43.498262Z
17
18 Registry #2: 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
19   :746cf728-5192-4266-9903-2ffb4b52cc5a
20   State: active
21   Max Credentials: 100
22   Issuer DID: did:indy:9pbXiFBZZGwXKp61HQBz3J
23   Tails Hash: 47xo7WNa2sbxHfnUkenzLgT5LotNci9Z1M5Dsp9vgnAN
24   Tails Location: /home/aries/.indy_client/tails/9pbXiFBZZGwXKp61HQBz3J:4:9
25     pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:746cf728-5192-4266-9903-2
26       ffb4b52cc5a/47xo7WNa2sbxHfnUkenzLgT5LotNci9Z1M5Dsp9vgnAN
27   Created: 2025-12-08T13:03:49.508221Z
28
29 =====
```

Listing A-71

Jupyter Notebook Cell 20

```
1 # Cell 20: Credential REVOKEN (staged)
2
3 print("="*60)
```

```

4 print("Credential REVOKEN (staged)")
5 print("=="*60 + "\n")
6
7 # Verwende gespeicherte Werte aus Cell 13
8 if 'rev_reg_id' in locals() and 'cred_rev_id' in locals():
9     print(f"Zu revokendes Credential:")
10    print(f"    Rev Reg ID: {rev_reg_id}")
11    print(f"    Cred Rev ID: {cred_rev_id}\n")
12
13 revoke_request = {
14     "rev_reg_id": rev_reg_id,
15     "cred_rev_id": cred_rev_id,
16     "notify": False, # Kein Notification-Webhook
17     "publish": False, # Staging only, publise später in Cell 21
18     "comment": "Revoked for testing purposes"
19 }
20
21 print("Senden Revocation Request (staging)...")
22 start_time = time.time()
23
24 revoke_response = api_post(
25     ISSUER_ADMIN_URL,
26     "/revocation/revoke",
27     revoke_request
28 )
29
30 if revoke_response is not None:
31     print(f"\nCredential erfolgreich REVOKED (staged)")
32     print(f"    Status: Pending (noch nicht auf Ledger)")
33     print(f"\nNächster Schritt: Führe Cell 22 aus um auf Ledger zu publishen!")
34 else:
35     print(f"\nRevocation fehlgeschlagen")
36     print(f"    Prüfe ob die IDs korrekt sind")
37 else:
38     print("Keine Revocation IDs gefunden!")
39     print("    Führe Cell 13 aus um Credential auszustellen")
40     print("    Die IDs werden automatisch in rev_reg_id und cred_rev_id gespeichert")
41
42 print("\n" + "=="*60)

```

Listing A-72

Jupyter Notebook Cell 20 Output

```

1 =====
2 Credential REVOKEN (staged)
3 =====
4
5 Zu revokendes Credential:
6     Rev Reg ID: 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
       :9efc6971-b713-465f-85db-a192b107b73e
7     Cred Rev ID: 1
8
9 Sende Revocation Request (staging)...
10
11 Credential erfolgreich REVOKED (staged)
12     Status: Pending (noch nicht auf Ledger)

```

```

13
14 Nächster Schritt: Führe Cell 21 aus um auf Ledger zu publishen!
15
16 =====

```

Listing A-73

Jupyter Notebook Cell 21

```

1 # Cell 21: Revocations auf Ledger PUBLISHEN
2
3 print("=="*60)
4 print("PUBLISH Revocations auf Ledger")
5 print("=="*60 + "\n")
6
7 print("Publishing alle pending Revocations auf Ledger...")
8
9 publish_response = api_post(
10     ISSUER_ADMIN_URL,
11     "/revocation/publish-revocations",
12     {}
13 )
14
15 if publish_response and "rrid2crid" in publish_response:
16     revoked_regs = publish_response["rrid2crid"]
17     total_revoked = sum(len(creds) for creds in revoked_regs.values())
18
19     print(f"\nERFOLGREICH auf Ledger published:")
20     print(f"    Revocation Registries: {len(revoked_regs)}")
21     print(f"    Total revoked Credentials: {total_revoked}")
22
23     for reg_id, cred_ids in revoked_regs.items():
24         print(f"        Registry: {reg_id}")
25         print(f"            Revoked Credential IDs: {cred_ids}")
26         print()
27
28     print("=="*60)
29     print("Revocations sind jetzt auf dem VON Ledger!")
30     print("    Holder kann nun KEINEN Non-Revocation-Proof mehr erstellen")
31     print("=="*60)
32 elif publish_response:
33     print(f"\nKeine pending Revocations zum Publishen")
34     print(f"    Response: {publish_response}")
35 else:
36     print(f"\nPublish fehlgeschlagen")
37
38 print("Jetzt erneut Cell 14 - 18 ausführen um Revocation zu testen!")
39
40 # =====
41 # Zeige Ledger-Transaktionen
42 # =====
43 print("\n" + "=="*60)
44 print("LEDGER-TRANSAKTIONEN")
45 print("=="*60)
46
47 try:
48     # Hole Domain Ledger Transaktionen

```

```

49     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
50
51 if ledger_response.status_code == 200:
52     ledger_data = ledger_response.json()
53     ledger_txns = ledger_data.get('results', [])
54
55     # Filtere nach seqNo 14
56     target_seqnos = [14]
57     found_txns = []
58
59     for txn in ledger_txns:
60         txn_metadata = txn.get('txnMetadata', {})
61         seqno = txn_metadata.get('seqNo')
62
63         if seqno in target_seqnos:
64             found_txns.append(txn)
65
66     # Sortiere nach seqNo aufsteigend
67     found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
68
69     if found_txns:
70         print(f"\n{n(len(found_txns)} Transaktionen gefunden:\n")
71
72         for txn in found_txns:
73             txn_metadata = txn.get('txnMetadata', {})
74             txn_data = txn.get('txn', {})
75             seqno = txn_metadata.get('seqNo')
76             txn_type = txn_data.get('type')
77             txn_time = txn_metadata.get('txnTime', 'N/A')
78
79             # Typ-Mapping
80             type_names = {
81                 '0' : 'NODE (Validator Node Registration)',
82                 '1' : 'NYM (DID Registration)',
83                 '4' : 'TXN_AUTHOR_AGREEMENT',
84                 '5' : 'TXN_AUTHOR_AGREEMENT_AML',
85                 '100': 'ATTRIB',
86                 '101': 'SCHEMA',
87                 '102': 'CRED_DEF',
88                 '112': 'CHANGE_KEY',
89                 '113': 'REVOC_REG_DEF',
90                 '114': 'REVOC_REG_ENTRY',
91                 '120': 'AUTH_RULE'
92             }
93             type_name = type_names.get(str(txn_type), f'Type {txn_type}')
94
95             print(f"====")
96             print(f"Seq No: {seqno} | Time: {txn_time}")
97             print(f"Transaction Type: {type_name}")
98             print(f"=====\n")
99
100            # Zeige vollständige Transaction
101            pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
102            print()
103
104        else:

```

```

105         print(f"\n NYM-Transaktion für DID (did_short) nicht gefunden")
106         print(f"    (Möglicherweise noch nicht im Cache)")
107     else:
108         print(f"\n Konnte Ledger nicht abrufen: {ledger_response.status_code}")
109
110 except Exception as e:
111     print(f"\n Fehler beim Abrufen der Ledger-Transaktion: {e}")
112
113 print("="#"*60)

```

Listing A-74*Jupyter Notebook Cell 21 Output*

```

1 =====
2 PUBLISH Revocations auf Ledger
3 =====
4
5 Publishing alle pending Revocations auf Ledger...
6
7 ERFOLGREICH auf Ledger published:
8     Revocation Registries: 1
9     Total revoked Credentials: 1
10    Registry: 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:9
11        efc6971-b713-465f-85db-a192b107b73e
12        Revoked Credential IDs: ['1']
13 =====
14 Revocations sind jetzt auf dem VON Ledger!
15     Holder kann nun KEINEN Non-Revocation-Proof mehr erstellen
16 =====
17 Jetzt erneut Cell 14 - 18 ausführen um Revocation zu testen!
18
19 =====
20 LEDGER-TRANSAKTIONEN
21 =====
22
23 1 Transaktionen gefunden:
24
25 =====
26 Seq No: 14 | Time: 1765203987
27 Transaction Type: REVOC_REG_ENTRY
28 =====
29
30
31 =====
32     Ledger Transaction (seqNo 14)
33 =====
34 {
35     "auditPath": [
36         "5YBFLtPW8BxW8JhBXAct4wLUhP53zefZY3svqsoq12K2",
37         "8vLEbJ5pzVFmhXYiddtoRA8iz7KA8KCANAKP2Xy2K7q",
38         "3edQ5ymPLwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP"
39     ],
40     "ledgerSize": 14,
41     "reqSignature": {
42         "type": "ED25519",

```


Listing A-75*Jupyter Notebook Cell 14 Output Revocation*

```

1 Holder Wallet - Gespeicherte Credentials:
2
3
4 =====
5 Credential #1
6 =====
7 Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
8 Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
9 Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
10 Revoked Status: True
11
12 Attribute:
13   - security_clearance_level: 2
14   - name: Mustermann
15   - facility_type: Umspannwerk Nord-Ost
16   - epoch_valid_from: 1765026000
17   - epoch_valid_until: 1765033200
18   - cert_type: Notfall-Wartungsberechtigung
19   - first_name: Max
20   - organisation: Musterfirma GmbH
21   - role: Notfalltechniker

```

Listing A-76*Jupyter Notebook Cell 15 Output Revocation*

```

1 Proof Request senden (Verifier --> Holder)...
2
3 Proof Request für KRITIS-Zutrittsberechtigung:
4   Endpoint:      /present-proof-2.0/send-request
5   Connection:    19df1913-7858-439c-9089-5a947b671c47
6   Cred Def ID:  9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
7   Proof Name:   Proof of KRITIS Emergency Maintenance Authorization
8   REVEALED:     Berechtigung, Anlage, Sicherheitsstufe, Gültigkeitszeitraum, Rolle
                  (6 Attribute)
9   DATENSCHUTZ: Identität (Vor-/Nachname) und Organisation werden NICHT offen gelegt (
                  Selective-Disclosure)
10
11 Proof Request gesendet
12   Exchange ID: e46cd865-d9b8-47c5-a551-258c1be4b82e
13   State:        request-sent

```

Listing A-77*Jupyter Notebook Cell 16 Output Revocation*

```

1 Holder: Proof Request verarbeiten...
2
3 Holder hat Proof Request empfangen
4   Presentation Exchange ID: e87389ed-d2a8-47f4-9f41-ef3309d2f62b
5
6 Verfügbare Credentials für Presentation:
7   Gefunden: 1 Credential(s)
8

```

```

9 Credential #1:
10 Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
11 Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintenance_cert:1.1
12 Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default
13 Attributes:
14   - cert_type: Notfall-Wartungsberechtigung
15   - name: Mustermann
16   - role: Notfalltechniker
17   - epoch_valid_from: 1765026000
18   - facility_type: Umspannwerk Nord-Ost
19   - first_name: Max
20   - organisation: Musterfirma GmbH
21   - epoch_valid_until: 1765033200
22   - security_clearance_level: 2
23
24 Total: 1 passende Credential(s) gefunden

```

Listing A-78

Jupyter Notebook Cell 17 Output Revocation

```

1 Holder: Presentation senden...
2
3 Credentials für Presentation gefunden
4 Format: Liste mit 1 Credential(s)
5
6 Mapped attribute 'attr3_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
7 Mapped attribute 'attr1_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
8 Mapped predicate 'pred1_clearance' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
9 Mapped attribute 'attr4_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
   c589757b65
10 Mapped attribute 'attr2_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
    c589757b65
11 Mapped attribute 'attr5_referent' --> Credential 39ac5fc4-efc2-45eb-9a21-01
    c589757b65
12
13 Presentation Mapping:
14   Requested Attributes: 5
15   Requested Predicates: 1
16   Total Mapped: 6
17
18 Sende Presentation...
19
20 Holder hat Presentation gesendet
21   Presentation Exchange ID: e87389ed-d2a8-47f4-9f41-ef3309d2f62b
22   State: presentation-sent
23   Role: prover
24
25 Warte auf Verification durch Verifier...
26
27 Finaler Holder Status: done
28 Presentation erfolgreich abgeschlossen!
29 Record bleibt verfügbar (--preserve-exchange-records aktiv)

```

Listing A-79*Jupyter Notebook Cell 18 Output Revocation*

```

1 Presentation verifizieren (Verifier)...
2
3 Proof erfolgreich verifiziert!
4
5 Verifizierte Attribute (REVEALED):
6   - epoch_valid_from: 1765026000
7   - epoch_valid_until: 1765033200
8   - facility_type: Umspannwerk Nord-Ost
9   - cert_type: Notfall-Wartungsberechtigung
10  - role: Notfalltechniker
11
12 PER ZKP VERIFIZIERT ABER DURCH DATENSCHUTZ GESCHÜTZT (UNREVEALED):
13  - Vorname: NICHT offengelegt (Zero-Knowledge-Proof)
14  - Nachname: NICHT offengelegt (Zero-Knowledge-Proof)
15  - Organisation: NICHT offengelegt (Zero-Knowledge-Proof)
16
17 State:      done
18 Verified:   false
19
20 Credential ist REVOKED!
21
22 =====
23 ZEITGÜLTIGKEITS-PRÜFUNG
24 =====
25 Aktueller Zeitpunkt: 2025-12-06 14:00:00 (Epoch: 1765029600) ==> Beispielwert
26 Gültig ab:          2025-12-06 13:00:00 (Epoch: 1765026000)
27 Gültig bis:         2025-12-06 15:00:00 (Epoch: 1765033200)
28
29 Zertifikat ist ZEITLICH GÜLTIG
30 =====
31
32 =====
33 ZERO-KNOWLEDGE-PROOF AUSWERTUNG
34 =====
35 Predicate erfüllt: security_clearance_level >= 2 (Ü2)
36 Zero-Knowledge-Proof: Exakte Sicherheitsstufe NICHT offengelegt!
37 Techniker hat mindestens Ü2 (Erweiterte Sicherheitsüberprüfung)
38 =====
39
40 =====
41 FINALE ZUGRIFFSENTSCHEIDUNG
42 =====
43
44 ZUGANG VERWEIGERT
45   Credential ist REVOKED
46
47 Zugang zum Umspannwerk Nord-Ost VERWEIGERT
48 =====
49
50 Privacy-Preserving Verification erfolgreich (DSGVO-konform)
```

Listing A-80*Jupyter Notebook Cell 22*

```

1 # Cell 22: Erst Zelle 14-18 ausführen... Credential Deletion - Holder löscht
2     Credential aus Wallet
3
4
5 # =====
6 # VORHER: Zeige alle Credentials
7 # =====
8 print("=="*60)
9 print("HOLDER WALLET - CREDENTIALS VORHER")
10 print("=="*60)
11
12 holder_creds_before = api_get(HOLDER_ADMIN_URL, "/credentials")
13
14 credentials_to_delete = []
15
16 if holder_creds_before and "results" in holder_creds_before and len(
17     holder_creds_before["results"]) > 0:
18     print(f"\n Holder hat {len(holder_creds_before['results'])} Credential(s)\n")
19
20     for idx, cred in enumerate(holder_creds_before["results"], 1):
21         referent = cred.get('referent', 'N/A')
22         schema_id = cred.get('schema_id', 'N/A')
23         cred_def_id = cred.get('cred_def_id', 'N/A')
24
25         print(f"Credential #{idx}:")
26         print(f"    Referent: {referent}")
27         print(f"    Schema ID: {schema_id[:50]}...")
28         print(f"    Cred Def ID: {cred_def_id[:50]}...")
29
30         # Revoked Status prüfen
31         revoked_response = api_get(HOLDER_ADMIN_URL, f'/credential/revoked/{referent}')
32         revoked_status = revoked_response.get('revoked', 'N/A') if revoked_response else
33             'N/A'
34         print(f"    Revoked: {revoked_status}")
35
36         # Attribute anzeigen
37         if "attrs" in cred:
38             print(f"    Attribute:")
39             for attr_name, attr_value in list(cred["attrs"].items()):
40                 print(f"        - {attr_name}: {attr_value}")
41
42         # Speichere Referent für Deletion
43         credentials_to_delete.append(referent)
44         print()
45     else:
46         print(" Holder hat noch keine Credentials im Wallet")
47         print(" Führe zuerst Cell 14-18 aus (Schema --> Cred Def --> Issuance)")
48
49 # =====
50 # DELETION: Lösche alle Credentials
51 # =====

```

```

52| if len(credentials_to_delete) > 0:
53|     print("\n  CREDENTIAL DELETION STARTEN\n")
54|
55| deleted_count = 0
56| failed_count = 0
57|
58| for idx, credential_id in enumerate(credentials_to_delete, 1):
59|     print(f"  Lösche Credential #{idx} (Referent: {credential_id[:30]}...)")
60|
61|     # API: DELETE /credential/{credential_id}
62|     delete_response = api_delete(
63|         HOLDER_ADMIN_URL,
64|         f"/credential/{credential_id}"
65|     )
66|
67|     if delete_response is not None or delete_response == {}:
68|         print(f"    Credential gelöscht")
69|         deleted_count += 1
70|     else:
71|         print(f"    Fehler beim Löschen")
72|         failed_count += 1
73|     print()
74|
75|     print("-" * 60)
76|     print(f"  DELETION SUMMARY")
77|     print("-" * 60)
78|     print(f"  Gelöscht: {deleted_count}/{len(credentials_to_delete)}")
79|     if failed_count > 0:
80|         print(f"  Fehler: {failed_count}")
81|     print("-" * 60)
82| else:
83|     print("\n  Keine Credentials zum Löschen vorhanden")
84|
85| # =====
86| # NACHHER: Zeige verbleibende Credentials
87| # =====
88| print("\n" + "=" * 60)
89| print("  HOLDER WALLET - CREDENTIALS NACHHER")
90| print("-" * 60)
91|
92| holder_creds_after = api_get(HOLDER_ADMIN_URL, "/credentials")
93|
94| if holder_creds_after and "results" in holder_creds_after and len(holder_creds_after["results"]) > 0:
95|     print(f"\n Holder hat noch {len(holder_creds_after['results'])} Credential(s)\n")
96|
97|     for idx, cred in enumerate(holder_creds_after["results"], 1):
98|         referent = cred.get('referent', 'N/A')
99|         schema_id = cred.get('schema_id', 'N/A')
100|        cred_def_id = cred.get('cred_def_id', 'N/A')
101|
102|        print(f"  Credential #{idx}:")
103|        print(f"    Referent: {referent}")
104|        print(f"    Schema ID: {schema_id[:50]}...")
105|        print(f"    Cred Def ID: {cred_def_id[:50]}...")
106|
107|        # Revoked Status prüfen

```

```

108     revoked_response = api_get(HOLDER_ADMIN_URL, f'/credential/revoked/{referent}')
109     revoked_status = revoked_response.get('revoked', 'N/A') if revoked_response
110     else 'N/A'
111     print(f"  Revoked: {revoked_status}")
112
113     # Attribute anzeigen
114     if "attrs" in cred:
115         print(f"    Attribute:")
116         for attr_name, attr_value in list(cred["attrs"].items()):
117             print(f"      - {attr_name}: {attr_value}")
118     print()
119 else:
120     print("\n Holder Wallet ist jetzt LEER (alle Credentials gelöscht)")
121 print("="*60)
122
123 print("\nHinweis:")
124 print("  Das Löschen eines Credentials aus dem Holder Wallet bedeutet:")
125 print("  - Credential ist LOKAL im Wallet gelöscht")
126 print("  - Credential ist NICHT auf dem Ledger revoked")
127 print("  - Issuer kann das Credential weiterhin sehen (--preserve-exchange-records")
128 print("  - Für echte Revocation: Siehe Cell 32-34 (Revocation Workflow)")
```

Listing A-81

Jupyter Notebook Cell 22 Output

```

1 Credential Deletion - Holder löscht Credential aus Wallet
2
3 =====
4 HOLDER WALLET - CREDENTIALS VORHER
5 =====
6
7 Holder hat 1 Credential(s)
8
9 Credential #1:
10    Referent: 39ac5fc4-efc2-45eb-9a21-01c589757b65
11    Schema ID: 9pbXiFBZZGwXKp61HQBz3J:2:kritis_emergency_maintena...
12    Cred Def ID: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default...
13    Revoked: True
14    Attribute:
15        - role: Notfalltechniker
16        - organisation: Musterfirma GmbH
17        - cert_type: Notfall-Wartungsberechtigung
18        - name: Mustermann
19        - facility_type: Umspannwerk Nord-Ost
20        - epoch_valid_until: 1765033200
21        - security_clearance_level: 2
22        - epoch_valid_from: 1765026000
23        - first_name: Max
24
25 =====
26
27 CREDENTIAL DELETION STARTEN
28
```

```

29| Lösche Credential #1 (Referent: 39ac5fc4-efc2-45eb-9a21-01c589...)
30|   Credential gelöscht
31|
32=====DELETION SUMMARY=====
33| DELETION SUMMARY
34| =====
35|   Gelöscht: 1/1
36| =====
37|
38=====HOLDER WALLET - CREDENTIALS NACHHER=====
39| HOLDER WALLET - CREDENTIALS NACHHER
40| =====
41|
42| Holder Wallet ist jetzt LEER (alle Credentials gelöscht)
43| =====
44|
45| Hinweis:
46|   Das Löschen eines Credentials aus dem Holder Wallet bedeutet:
47|   - Credential ist LOKAL im Wallet gelöscht
48|   - Credential ist NICHT auf dem Ledger revoked
49|   - Issuer kann das Credential weiterhin sehen (--preserve-exchange-records)
50|   - Für echte Revocation: Siehe Cell 32-34 (Revocation Workflow)

```

Listing A-82

Jupyter Notebook Cell 23

```

1| # Cell 23: Revocation Registry ROTATION (Optional)
2|
3| print("=".*60)
4| print("Revocation Registry ROTATION")
5| print("=".*60 + "\n")
6|
7| print("Registry Rotation ist nötig wenn:")
8| print("  Registry voll ist (100 Credentials mit TAILS_FILE_COUNT=100)")
9| print("  Neue Registry für weitere Credentials benötigt wird\n")
10|
11| if 'cred_def_id' in locals() and cred_def_id:
12|     print(f"Rotiere Registry für Cred Def: {cred_def_id[:50]}...\n")
13|
14| start_time = time.time()
15|
16| rotate_response = api_post(
17|     ISSUER_ADMIN_URL,
18|     f"/revocation/active-registry/{cred_def_id}/rotate",
19|     {}
20| )
21|
22| duration = time.time() - start_time
23|
24| if rotate_response and "rev_reg_ids" in rotate_response:
25|     decom = rotate_response["rev_reg_ids"]
26|     print(f"Registry erfolgreich rotiert!")
27|     print(f"  Decommissioned Registries: {len(decom)}")
28|     for reg_id in decom:
29|         print(f"    - {reg_id[:50]}...")
30|     print(f"  Zeit: {duration:.3f}s")

```

```

31     print(f"\n  Neue aktive Registry wurde erstellt")
32     print(f"  Weitere Credentials können jetzt ausgestellt werden")
33 else:
34     print(f"Rotation nicht möglich")
35     print(f"  Möglicherweise ist die Registry noch nicht voll")
36     print(f"  Oder es gibt keine aktive Registry für diese Cred Def")
37 if rotate_response:
38     print(f"  Response: {rotate_response}")
39 else:
40     print("Keine Credential Definition ID gefunden")
41     print("  Führe Cell 7 aus um Cred Def zu erstellen")
42
43 # =====
44 # Zeige Ledger-Transaktionen
45 # =====
46 print("\n" + "="*60)
47 print("LEDGER-TRANSAKTIONEN")
48 print("="*60)
49
50 try:
51     # Hole Domain Ledger Transaktionen
52     ledger_response = requests.get(f"{VON_NETWORK_URL}/ledger/domain")
53
54     if ledger_response.status_code == 200:
55         ledger_data = ledger_response.json()
56         ledger_txns = ledger_data.get('results', [])
57
58         # Filtere nach seqNo 15-18
59         target_seqnos = [15,16,17,18]
60         found_txns = []
61
62         for txn in ledger_txns:
63             txn_metadata = txn.get('txnMetadata', {})
64             seqno = txn_metadata.get('seqNo')
65
66             if seqno in target_seqnos:
67                 found_txns.append(txn)
68
69             # Sortiere nach seqNo aufsteigend
70             found_txns.sort(key=lambda x: x.get('txnMetadata', {}).get('seqNo', 0))
71
72         if found_txns:
73             print(f"\n{len(found_txns)} Transaktionen gefunden:\n")
74
75         for txn in found_txns:
76             txn_metadata = txn.get('txnMetadata', {})
77             txn_data = txn.get('txn', {})
78             seqno = txn_metadata.get('seqNo')
79             txn_type = txn_data.get('type')
80             txn_time = txn_metadata.get('txnTime', 'N/A')
81
82             # Typ-Mapping
83             type_names = {
84                 '0' : 'NODE (Validator Node Registration)',
85                 '1' : 'NYM (DID Registration)',
86                 '4' : 'TXN_AUTHOR AGREEMENT',
87                 '5' : 'TXN_AUTHOR AGREEMENT_AML',

```

```

88         '100': 'ATTRIB',
89         '101': 'SCHEMA',
90         '102': 'CRED_DEF',
91         '112': 'CHANGE_KEY',
92         '113': 'REVOC_REG_DEF',
93         '114': 'REVOC_REG_ENTRY',
94         '120': 'AUTH_RULE'
95     }
96     type_name = type_names.get(str(txn_type), f'Type {txn_type}')
97
98     print(f"====")
99     print(f"Seq No: {seqno} | Time: {txn_time}")
100    print(f"Transaction Type: {type_name}")
101    print(f"====\n")
102
103    # Zeige vollständige Transaction
104    pretty_print(txn, f'Leder Transaction (seqNo {seqno})')
105    print()
106
107    else:
108        print(f'\n  NYM-Transaktion für DID {did_short} nicht gefunden')
109        print(f"  (Möglichlicherweise noch nicht im Cache)")
110    else:
111        print(f'\n  Konnte Leder nicht abrufen: {ledger_response.status_code}')
112
113 except Exception as e:
114     print(f'\n  Fehler beim Abrufen der Leder-Transaktion: {e}')
115
116 print("=="*60)

```

Listing A-83

Jupyter Notebook Cell 23 Output

```

1 =====
2 Revocation Registry ROTATION
3 =====
4
5 Registry Rotation ist nötig wenn:
6   Registry voll ist (100 Credentials mit TAILS_FILE_COUNT=100)
7   Neue Registry für weitere Credentials benötigt wird
8
9 Rotiere Registry für Cred Def: 9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default...
10
11 Registry erfolgreich rotiert!
12   Decommissioned Registries: 2
13     - 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:...
14     - 9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:...
15   Zeit: 9.240s
16
17   Neue aktive Registry wurde erstellt
18   Weitere Credentials können jetzt ausgestellt werden
19
20 =====
21 LEDGER-TRANSAKTIONEN
22 =====

```

```

23
24 4 Transaktionen gefunden:
25
26 =====
27 Seq No: 15 | Time: 1765204680
28 Transaction Type: REVOC_REG_DEF
29 =====
30
31
32 =====
33   Ledger Transaction (seqNo 15)
34 =====
35 {
36   "auditPath": [
37     "DYTvAiJWJquEG4T6nczCMN5cDmhQnwGueDpD1xac18cR",
38     "EMGMNnNpvSpg7QReuEGGpeH5PmGacDcmfxQJFHnnPk3j",
39     "8vLEbJ5pzVFmhXYiddtoRA8iz7KA8KCANAOKP2Xy2K7q",
40     "3edQ5ymPLwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP",
41     "EXMbifQNEjrLsthAMg6fV1cMyYecd7Zq6X9XnJxmT69D"
42   ],
43   "ledgerSize": 18,
44   "reqSignature": {
45     "type": "ED25519",
46     "values": [
47       {
48         "from": "9pbXiFBZZGwXKp61HQBz3J",
49         "value": "4UtWmZaPxLan3Bd1bj1dyC2oHAFW4UZ6afzxuEQKas
50           PfZvBdPf7lshyUGc3asTDtdBaxK8iEohD1c5duzaRrV9Wb"
51       }
52     ]
53   },
54   "rootHash": "Ds9kSEoz9MyFezpb7VzUGxW6UcCeViewHkP3kcywNMRK",
55   "txns": {
56     "data": {
57       "credDefId": "9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default",
58       "id": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:
59         f4627aa8-cd65-4d82-933d-3926a0c8dfbf",
60       "revocDefType": "CL_ACCUM",
61       "tag": "f4627aa8-cd65-4d82-933d-3926a0c8dfbf",
62       "value": {
63         "issuanceType": "ISSUANCE_BY_DEFAULT",
64         "maxCredNum": 100,
65         "publicKeys": {
66           "accumKey": {
67             "z": "1 04DE04E87D0BDE75DFB33D3E50068E355BA1EC188B6D842F3EFFE7C1AF0BC9CF 1
68               039F921C3145E6E8398446E8D5EE70984B65C311C4983807569DD7A8C46C643F 1 1
69               D663B4362C20BF47EF4B241309D68BFC2FE601884D899B0C24ADB8DB005CD54 1 2116
70               CBF8301A5153CB6C7101A6DC6A26EE1857A5B192A4948FC63F7C2371284D 1 118054
71               A8CA7BF15588D913C51C59DB368B71D007323DFD6517DB5978CA8ADF4F 1 1
72               A59E88D132973B0F820EF157BF3F1AEAE0CB7E79EB193BC637A26DC4EAA3B62 1 20
73               F1CC03F7562924321AF23336877D138FE046B4CEFA667BC20ED2F4B673275D 1 00
74               F1E196990CCE6AEF2CADA00E59B421D8091B277651FB061DBD8F5C58418C00 1 07
75               A3A63EDDCFFEB6A8F66C1B48F32FCA31CDAB982F8778CED5322A28BB0815D3 1 1235
76               D079A86E966BF1DE7E3CDBEE2EE106090EB34991350C62F312DC7341B269 1 00
77               C0C58D37DC4FC9964322C1A2B80EF43202B80C1FF24DA3CF6F72593EBD52F8 1 1363
78               C943EF8C33FF2C0EF30547CDD8BCA6DFD94FEE4C05DF655F6750B21FF1D7"
79         }
80       }
81     }
82   }
83 }
```

```

68      },
69      "tailsHash": "3WUqN4BfHNpg5fG3Kv7BScb6jX83u178qpDWwix1CqSD",
70      "tailsLocation": "https://host.docker.internal:6543/9pbXiFBZZGwXKp61HQBz3J:4:9
    pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:f4627aa8-cd65-4d82-933d-3926
        a0c8dfbf"
71  },
72 },
73 "metadata": {
74   "digest": "98c20262e889aee4c4a671b0809905b164557ba4b0e274dca8b56e8031b21807",
75   "from": "9pbXiFBZZGwXKp61HQBz3J",
76   "payloadDigest": "
        fa828434660d5cbeb4f5075dff26d8952125935ebd26bdf6ace85e2a15df5722",
77   "reqId": 1765204680753177624
78 },
79   "protocolVersion": 2,
80   "type": "113"
81 },
82 "txnMetadata": {
83   "seqNo": 15,
84   "txnid": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:
        f4627aa8-cd65-4d82-933d-3926a0c8dfbf",
85   "txnTime": 1765204680
86 },
87   "ver": "1"
88 }
89 =====
90 Seq No: 16 | Time: 1765204683
91 Transaction Type: REVOC_REG_ENTRY
92 =====
93 =====
94 =====
95 =====
96 =====
97 Ledger Transaction (seqNo 16)
98 =====
99 {
100   "auditPath": [
101     "22mQjUSSjktyj47kdfBTJymGGZvnWGwa4kyjtF7K1PcR",
102     "EMGMNnNpvSpg7QReuEGGpeH5PmGacDcmfxQJFHnnPk3j",
103     "8vLEbJ5pzVFmhXYiddtoRA8iz7KA8KCANAkNP2Xy2K7q",
104     "3edQ5ymlPlwVpBdhaYGETeewRj1X48jeuZxzHnoDYCgNP",
105     "EXMbfQNEjrLsthAMg6fV1cMyYecd7Zq6X9XnJxmT69D"
106   ],
107   "ledgerSize": 18,
108   "reqSignature": {
109     "type": "ED25519",
110     "values": [
111       {
112         "from": "9pbXiFBZZGwXKp61HQBz3J",
113         "value": "4TgYaZSqivGmmpM3hxytd2N92gtPY8kEFiKh8F2KZA
            X9gnDwRZgTGZM3f7L9qTm7UBB5LBXGcvFBVkFtECYAihpn"
114       }
115     ]
116   },
117   "rootHash": "Ds9kSEoz9MyFezpb7VzUGxW6UcCeViewHkP3kcywNMRK",
118   "txin": {
119     "data": {

```



```

170 "txn": {
171   "data": {
172     "credDefId": "9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default",
173     "id": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
174       :742a69d5-a130-42e4-b052-3e4d9c80910b",
175     "revocDefType": "CL_ACCUM",
176     "tag": "742a69d5-a130-42e4-b052-3e4d9c80910b",
177     "value": {
178       "issuanceType": "ISSUANCE_BY_DEFAULT",
179       "maxCredNum": 100,
180       "publicKeys": {
181         "accumKey": {
182           "z": "1 242AC12CD4E32554A453156CF77DDEE84375F1B9491D8D0DDF19D55D5C7803E 1
183             20579E13429F5C578DC66E0E3AF1C86B38A17E2BA7EF7647E6534056DBEAA832 1 1
184             E769B6633DB9FF6293E55551F6E22A59351F416CB95F194BC7ECA4C008FE660 1 20
185             E8FE1746CCBAB6B66E2F1CFD629C5FEDC872AAE116A39CB18F0965CBE24DD6 1
186             201411E1DD59576410D5EA133EFB08EEA7E01210BCE71852D2D10E2112427C12 1 11
187             C0F13B16ADC12C769A4A66CB6FC601199084BD0D08E7D7EFDA2BFAF6B88B06 1 036
188             C3030B009C9B78D253BAF8D68DE13C85F952DF6BABC92BB28F19008A74B99 1 0686
189             D758926A30CBE9718B560F19FE6AE0FCBF6F202638EDD0ED14782E1F6778 1 09391
190             DC25B35D363BC70113BF5360BA9E01DD4F6783F4F40DDE490DC75ACF45F 1 07
191             A3A5CF83D42076BEE4C7351DE05E336BCBC188DD457519B0CF668E691919A9 1
192             1906826C9F44FB77D6F892CCC2DF56B814F86025D035C721FCA49D07DAE56695 1 1
193             B3056E40147E7282F4E7B5F5F225D7DF344EDD1044F95725C62900EA0C132C"
194           }
195         },
196       },
197     },
198     "tailsHash": "BwauSxtMkyPJbZhxAUvQos5WEEm3vnbrPyXASQcXoFwfT",
199     "tailsLocation": "https://host.docker.internal:6543/9pbXiFBZZGwXKp61HQBz3J:4:9
200       pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM:742a69d5-a130-42e4-b052-3
201       e4d9c80910b"
202   },
203   "metadata": {
204     "digest": "8ded22d128f2c9d934b943f16f279b79b5bbe739ff27e8513a22213cfe6c3de1",
205     "from": "9pbXiFBZZGwXKp61HQBz3J",
206     "payloadDigest": "
207       ada7b69a040cc9494a218b0d0e7385cb7e71b77f5ac9d7400f5d70b0b00e3419",
208     "reqId": 1765204683896845321
209   },
210   "protocolVersion": 2,
211   "type": "113"
212 },
213 "txnMetadata": {
214   "seqNo": 17,
215   "txnId": "9pbXiFBZZGwXKp61HQBz3J:4:9pbXiFBZZGwXKp61HQBz3J:3:CL:8:default:CL_ACCUM
216       :742a69d5-a130-42e4-b052-3e4d9c80910b",
217   "txnTime": 1765204686
218 },
219   "ver": "1"
220 }
221 =====
222 Seq No: 18 | Time: 1765204689
223 Transaction Type: REVOC_REG_ENTRY
224 =====
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2
```


Anhang 5.2: Validierung der KRITIS-Compliance-Anforderungen

Protokollierung Sicherheitsrelevanter Ereignisse

Listing A-84

Issuer acapy agent logs

```

1 2025-12-10 18:55:12,652 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:12 +0000] "GET /connections HTTP/1.0" 200 3425 "-" "python-requests/2.32.5"
2 2025-12-10 18:55:12,671 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:12 +0000] "GET /connections/d75e516d-66dd-473a-8aa4-ee417a85ef4e HTTP/1.0" 200 3410 "-" "python-requests/2.32.5"
3 2025-12-10 18:55:22,852 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:22 +0000] "POST /issue-credential-2.0/send-offer HTTP/1.0" 200 22208 "-" "python-requests/2.32.5"
4 2025-12-10 18:55:22,854 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO Using PQC pack mode: sender_pqc=True, recipients=['pqc']
5 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO Using PQC unpack mode: alg=PQC-Authcrypt
6 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO =====
7 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Starting recipient decryption loop
8 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Found 1 recipients in JWE
9 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Recipient KIDs: ['39zdn...sovF']
10 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Trying recipient: 39zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG...
11 2025-12-10 18:55:23,008 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Calling session.fetch_key(39 zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG...)
12 2025-12-10 18:55:23,009 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] fetch_key result: PQCKey(algorithm=ml-kem-768)
13 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Verifying ML-DSA-65 signature...
14 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] sender_vk (base58): 5sN6ZWoHgEv8dwS8z7MGrdhGoZvwsV8foHf2tPiT3E5DVrcr28ch5L9JK2JT ...
15 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] sender_public (bytes): 1952 bytes
16 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Signature verified!
17 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] SUCCESS! recip_vk set to: 39 zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG
18 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] Loop finished. recip_vk = 39zd...ovF
19 2025-12-10 18:55:23,013 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO =====
20 2025-12-10 18:55:23,014 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:23 +0000] "POST / HTTP/1.0" 200 75 "-" "Python/3.12 aiohttp/3.12.15"
21 2025-12-10 18:55:23,107 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO Using PQC pack mode: sender_pqc=True, recipients=['pqc']
22 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO Using PQC unpack mode: alg=PQC-Authcrypt

```

```

23| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO
=====
24| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Starting recipient decryption loop
25| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Found 1 recipients in JWE
26| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Recipient KIDs: ['39zdnQamP...TAsovF']
27| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Trying recipient: 39zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG...
28| 2025-12-10 18:55:23,250 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Calling session.fetch_key(39
    zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG...)
29| 2025-12-10 18:55:23,251 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    fetch_key result: PQCKey(algorithm=ml-kem-768)
30| 2025-12-10 18:55:23,254 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Verifying ML-DSA-65 signature...
31| 2025-12-10 18:55:23,254 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    sender_vk (base58): 5sN6ZWoHgEv8dwS8z7MGrdhGoZvwsV8foHf2tPiT3E5DVrscr28ch5L9JK2JT
    ...
32| 2025-12-10 18:55:23,254 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    sender_public (bytes): 1952 bytes
33| 2025-12-10 18:55:23,255 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG] ok
    Signature verified!
34| 2025-12-10 18:55:23,255 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    SUCCESS! recip_vk set to: 39
    zdnQamPoRAWZiAT3YAA3ns1XnqMi9TtictVeSJW215p5pQUKJWm5bvYekG
35| 2025-12-10 18:55:23,255 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO [PQC UNPACK DEBUG]
    Loop finished. recip_vk = 39zd...LEj9gsyymTAsovF
36| 2025-12-10 18:55:23,255 pqc_didpeer4_fm.v1_0.pqc_didcomm_v1 INFO
=====
37| 2025-12-10 18:55:23,256 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:23 +0000] "
    POST / HTTP/1.0" 200 75 "-" "Python/3.12 aiohttp/3.12.15"
38| 2025-12-10 18:55:27,891 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:27 +0000] "
    GET /issue-credential-2.0/records/90e02709-9ab2-4fc... HTTP/1.0"
    200 43641 "-" "python-requests/2.32.5"
39| 2025-12-10 18:55:27,937 aiohttp.access INFO 172.23.0.3 [10/Dec/2025:18:55:27 +0000] "
    GET /issue-credential-2.0/records HTTP/1.0" 200 43656 "-" "python-requests/2.32.5"

```

Listing A-85

Issuer Sidecar-Proxy logs

```

1 172.23.0.1 - - [10/Dec/2025:18:55:10 +0000] "POST /out-of-band/create-invitation?
    auto_accept=true&multi_use=false HTTP/1.1" 200 16180 "-" "python-requests/2.32.5"
    "-"
2 2025/12/10 18:55:10 [info] 8#0: *23 client 172.23.0.1 closed keepalive connection
3 2025/12/10 18:55:10 [warn] 8#0: *25 a client request body is buffered to a temporary
    file /opt/nginx/client_body_temp/0000000001, client: 172.23.0.1, server: pqc-
    sidecarproxy-issuer, request: "POST / HTTP/1.1", host: "host.docker.internal:8020"
4 172.23.0.1 - - [10/Dec/2025:18:55:10 +0000] "POST / HTTP/1.1" 200 0 "-" "Python/3.12
    aiohttp/3.12.15" "-"
5 2025/12/10 18:55:10 [warn] 8#0: *25 a client request body is buffered to a temporary
    file /opt/nginx/client_body_temp/0000000002, client: 172.23.0.1, server: pqc-
    sidecarproxy-issuer, request: "POST / HTTP/1.1", host: "host.docker.internal:8020"
6 172.23.0.1 - - [10/Dec/2025:18:55:10 +0000] "POST / HTTP/1.1" 200 0 "-" "Python/3.12
    aiohttp/3.12.15" "-"

```

```

7| 2025/12/10 18:55:11 [warn] 8#0: *25 a client request body is buffered to a temporary
8|   file /opt/nginx/client_body_temp/0000000003, client: 172.23.0.1, server: pqc-
9|   sidecarproxy-issuer, request: "POST / HTTP/1.1", host: "host.docker.internal:8020"
10| 172.23.0.1 - - [10/Dec/2025:18:55:11 +0000] "POST / HTTP/1.1" 200 0 "-" "Python/3.12
11|   aiohttp/3.12.15" "-"
12| 172.23.0.1 - - [10/Dec/2025:18:55:12 +0000] "GET /connections HTTP/1.1" 200 3300 "-" "python-requests/2.32.5" "-"
13| 2025/12/10 18:55:12 [info] 8#0: *29 client 172.23.0.1 closed keepalive connection
14| 172.23.0.1 - - [10/Dec/2025:18:55:12 +0000] "GET /connections/d75e516d-66dd-473a-8aa4-
15|   ee417a85ef4e HTTP/1.1" 200 3285 "-" "python-requests/2.32.5" "-"
16| 2025/12/10 18:55:12 [info] 8#0: *31 client 172.23.0.1 closed keepalive connection
17| 172.23.0.1 - - [10/Dec/2025:18:55:22 +0000] "POST /issue-credential-2.0/send-offer
18|   HTTP/1.1" 200 22082 "-" "python-requests/2.32.5" "-"
19| 2025/12/10 18:55:22 [info] 8#0: *33 client 172.23.0.1 closed keepalive connection
20| 2025/12/10 18:55:23 [warn] 8#0: *25 a client request body is buffered to a temporary
21|   file /opt/nginx/client_body_temp/0000000004, client: 172.23.0.1, server: pqc-
22|   sidecarproxy-issuer, request: "POST / HTTP/1.1", host: "host.docker.internal:8020"
23| 172.23.0.1 - - [10/Dec/2025:18:55:23 +0000] "POST / HTTP/1.1" 200 0 "-" "Python/3.12
24|   aiohttp/3.12.15" "-"
25| 2025/12/10 18:55:23 [warn] 8#0: *25 a client request body is buffered to a temporary
26|   file /opt/nginx/client_body_temp/0000000005, client: 172.23.0.1, server: pqc-
27|   sidecarproxy-issuer, request: "POST / HTTP/1.1", host: "host.docker.internal:8020"
28| 172.23.0.1 - - [10/Dec/2025:18:55:23 +0000] "POST / HTTP/1.1" 200 0 "-" "Python/3.12
29|   aiohttp/3.12.15" "-"
30| 172.23.0.1 - - [10/Dec/2025:18:55:27 +0000] "GET /issue-credential-2.0/records/90
31|   e02709-9ab2-4fca-b2e4-bb43d51f2191 HTTP/1.1" 200 43515 "-" "python-requests/2.32.5
32|   " "-"
33| 2025/12/10 18:55:46 [info] 11#0: *102 SSL_do_handshake() failed (SSL: error:0A000102:
34|   SSL routines::unsupported protocol) while SSL handshaking, client: 172.23.0.1,
35|   server: 0.0.0.0:8021

```

Anhang 5.3: Validierung der Kryptoagilität - Applikationslayer

Listing A-86

Jupyter Notebook Cell 9 Demonstration Kryptoagilität ed25519

```

1 # Cell 9: Connection Issuer Holder erstellen
2
3 print("Connection: Issuer --> Holder erstellen...\n")
4
5 start_time = time.time()
6
7 # =====
8 # 1. PRE-CHECK: Existierende Connections prüfen
9 # =====
10 print("Pre-Check: Prüfe existierende Connections...\n")
11
12 existing_conn_issuer_holder = None
13 existing_conn_holder = None
14
15 # Prüfe Issuer-Seite: GET /connections
16 issuer_conns_response = api_get(ISSUER_ADMIN_URL, "/connections")
17 if issuer_conns_response and "results" in issuer_conns_response:

```

```

18     print(f"    Issuer: {len(issuer_conns_response['results'])} Connection(s) gefunden")
19     )
20     for conn in issuer_conns_response["results"]:
21         if conn.get("state") == "completed":
22             existing_conn_issuer_holder = conn["connection_id"]
23             print(f"        Connection {existing_conn_issuer_holder} (State: completed)")
24             break
25     else:
26         print("    Issuer: Keine Connections gefunden")
27
28 # Prüfe Holder-Seite: GET /connections
29 holder_conns_response = api_get(HOLDER_ADMIN_URL, "/connections")
30 if holder_conns_response and "results" in holder_conns_response:
31     print(f"    Holder: {len(holder_conns_response['results'])} Connection(s) gefunden")
32     )
33     for conn in holder_conns_response["results"]:
34         if conn.get("state") == "completed":
35             existing_conn_holder = conn["connection_id"]
36             print(f"        Connection {existing_conn_holder} (State: completed)")
37             break
38     else:
39         print("    Holder: Keine Connections gefunden")
40
41 if existing_conn_issuer_holder and existing_conn_holder:
42     print(f"\nExistierende Connection gefunden!")
43     print(f"    Issuer Connection ID: {existing_conn_issuer_holder}")
44     print(f"    Holder Connection ID: {existing_conn_holder}")
45     print("    Überspringe Connection-Erstellung\n")
46     conn_id_issuer_holder = existing_conn_issuer_holder
47     conn_id_holder = existing_conn_holder
48 else:
49     print("\n    Keine vollständige existierende Connection gefunden")
50     print("    Erstelle neue Connection mit did:peer:4...\n")
51
52 # =====
53 # 2. HAUPT-WORKFLOW: Connection mit did:peer erstellen
54 # =====
55
56 # Initialisiere Variablen
57 conn_id_issuer_holder = None
58 conn_id_holder = None
59 invitation_msg_id = None
60 invitation_key = None
61
62 # Issuer erstellt Out-of-Band Invitation mit did:peer:4
63 invitation_data = {
64     "handshake_protocols": ["https://didcomm.org/didexchange/1.1"],
65     "use_did_method": "did:peer:4",
66     "metadata": {"key_type": "ed25519"},
67     "my_label": "Issuer Agent",
68     "goal": "Establish connection for credential issuance",
69     "goal_code": "issue-vc",
70     "accept": [
71         "didcomm/aip1",
72         "didcomm/aip2;env=rfc19"
73     ]
74 }

```

```

73
74     print("Invitation-Modus: did:peer:4 (P2P Connection)")
75
76     # API Call mit Query-Parametern für auto-accept
77     invitation_response = api_post(
78         ISSUER_ADMIN_URL,
79         "/out-of-band/create-invitation?auto_accept=true&multi_use=false",
80         invitation_data
81     )
82
83     if invitation_response and "invitation" in invitation_response:
84         invitation = invitation_response["invitation"]
85         oob_id = invitation_response.get("oob_id")
86         invitation_url = invitation_response.get("invitation_url", "")
87         state = invitation_response.get("state")
88
89         # WICHTIG: Speichere invitation_msg_id aus der Invitation
90         invitation_msg_id = invitation.get("@id")
91
92         # Optional: Extrahiere invitation_key aus services (falls vorhanden)
93         services = invitation.get("services", [])
94         if services and len(services) > 0:
95             # Bei did:peer:4 ist der Service oft inline
96             service = services[0]
97             if isinstance(service, dict):
98                 recipient_keys = service.get("recipientKeys", [])
99                 if recipient_keys:
100                     invitation_key = recipient_keys[0]
101
102         print(f"Issuer: Invitation erstellt")
103         print(f"    OOB ID: {oob_id}")
104         print(f"    State: {state}")
105         print(f"    Invitation Msg ID: {invitation_msg_id}")
106         if invitation_key:
107             print(f"    Invitation Key: {invitation_key[:30]}...")
108         print(f"    Services: {len(invitation.get('services', []))} service(s)")
109         print(f"    Label: {invitation_data['my_label']}")
110         print(f"    Goal: {invitation_data['goal']}")
111
112         # Holder akzeptiert Invitation (mit auto-accept)
113         receive_response = api_post(
114             HOLDER_ADMIN_URL,
115             "/out-of-band/receive-invitation?auto_accept=true",
116             invitation
117         )
118
119         if receive_response is not None:
120             conn_id_holder = receive_response.get("connection_id")
121             print(f"\nHolder: Invitation akzeptiert")
122             print(f"    Connection ID (Holder): {conn_id_holder}")
123
124             # Issuer findet seine Connection via invitation_msg_id
125             print(f"\n    Issuer sucht Connection via invitation_msg_id...")
126             time.sleep(2)  # Kurze Wartezeit für DIDExchange Protocol
127
128             # Hole alle Issuer Connections
129             issuer_conns = api_get(ISSUER_ADMIN_URL, "/connections")

```

```

130         if issuer_conns and "results" in issuer_conns:
131             connections = issuer_conns["results"]
132
133             print(f"      Prüfe {len(connections)} Connection(s)...")
134
135             # Finde Connection anhand invitation_msg_id (exaktes Match!)
136             connection_found = False
137             for conn in connections:
138                 conn_invitation_msg_id = conn.get("invitation_msg_id")
139                 conn_invitation_key = conn.get("invitation_key")
140
141                 # Match via invitation_msg_id (bevorzugt)
142                 if invitation_msg_id and conn_invitation_msg_id == invitation_msg_id:
143                     conn_id_issuer_holder = conn["connection_id"]
144                     conn_state = conn.get("state")
145                     print(f"\n      Issuer: Connection gefunden via invitation_msg_id!")
146                     print(f"      Connection ID: {conn_id_issuer_holder}")
147                     print(f"      State: {conn_state}")
148                     print(f"      Invitation Msg ID: {conn_invitation_msg_id}")
149                     print(f"      Their Label: {conn.get('their_label', 'N/A')}")
150                     print(f"      Updated At: {conn.get('updated_at', 'N/A')}")
151                     connection_found = True
152                     break
153
154             # Fallback: Match via invitation_key
155             elif invitation_key and conn_invitation_key == invitation_key:
156                 conn_id_issuer_holder = conn["connection_id"]
157                 conn_state = conn.get("state")
158                 print(f"\n      Issuer: Connection gefunden via invitation_key!")
159                 print(f"      Connection ID: {conn_id_issuer_holder}")
160                 print(f"      State: {conn_state}")
161                 print(f"      Invitation Key: {conn_invitation_key[:30]}...")
162                 print(f"      Their Label: {conn.get('their_label', 'N/A')}")
163                 connection_found = True
164                 break
165
166             if not connection_found:
167                 print(f"      Keine Connection mit matching invitation_msg_id gefunden")
168                 print(f"      Gesuchte invitation_msg_id: {invitation_msg_id}")
169             else:
170                 print(f"      Konnte Issuer Connections nicht abrufen")
171             else:
172                 print("Holder konnte Invitation nicht akzeptieren")
173             else:
174                 print("Fehler beim Erstellen der Invitation")
175
176 # =====
177 # 3. POST-VALIDATION: Connection-Status verifizieren
178 # =====
179 print("\nPost-Validation: Verifizierte Connection-Status...\n")
180
181 validation_success = False
182 issuer_state = None
183 holder_state = None

```

```

184
185 if conn_id_issuer_holder:
186     # Prüfe Issuer-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
187     issuer_conn_detail = api_get(
188         ISSUER_ADMIN_URL,
189         f"/connections/{conn_id_issuer_holder}"
190     )
191
192     if issuer_conn_detail:
193         issuer_state = issuer_conn_detail.get("state")
194         issuer_their_label = issuer_conn_detail.get("their_label")
195         issuer_their_role = issuer_conn_detail.get("their_role")
196         issuer_invitation_msg_id = issuer_conn_detail.get("invitation_msg_id")
197
198         print(f"    Issuer Connection {conn_id_issuer_holder}:")
199         print(f"        State: {issuer_state}")
200         print(f"        Their Label: {issuer_their_label}")
201         print(f"        Their Role: {issuer_their_role}")
202         print(f"        Invitation Msg ID: {issuer_invitation_msg_id}")
203
204     # Prüfe Holder-Seite (falls conn_id_holder bekannt)
205     if conn_id_holder:
206         # Prüfe Holder-Seite mit spezifischem Endpoint: GET /connections/{conn_id}
207         holder_conn_detail = api_get(
208             HOLDER_ADMIN_URL,
209             f"/connections/{conn_id_holder}"
210         )
211
212         if holder_conn_detail:
213             holder_state = holder_conn_detail.get("state")
214             holder_their_label = holder_conn_detail.get("their_label")
215             holder_their_role = holder_conn_detail.get("their_role")
216             holder_invitation_msg_id = holder_conn_detail.get("invitation_msg_id")
217
218             print(f"\n    Holder Connection {conn_id_holder}:")
219             print(f"        State: {holder_state}")
220             print(f"        Their Label: {holder_their_label}")
221             print(f"        Their Role: {holder_their_role}")
222             print(f"        Invitation Msg ID: {holder_invitation_msg_id}")
223
224         # Verifiziere dass invitation_msg_id übereinstimmt
225         if invitation_msg_id and issuer_invitation_msg_id == holder_invitation_msg_id:
226             print(f"        Invitation Msg IDs stimmen überein!")
227
228     # Validierung
229     if issuer_state == "active" and holder_state == "active":
230         validation_success = True
231     elif issuer_state == "active" and not conn_id_holder:
232         # Bei existierender Connection nur Issuer-Seite bekannt
233         validation_success = True
234
235     # Finale Ausgabe
236     print(f"\n{'='*60}")
237     if validation_success:
238         print(f"CONNECTION ESTABLISHED (did:peer:4): Issuer <--> Holder")
239         print(f"    Issuer Connection ID: {conn_id_issuer_holder}")
240         if conn_id_holder:

```

```

241     print(f"    Holder Connection ID: {conn_id_holder}")
242     print(f"    Issuer State: {issuer_state}")
243     if holder_state:
244         print(f"    Holder State: {holder_state}")
245     if invitation_msg_id:
246         print(f"    Invitation Msg ID: {invitation_msg_id}")
247 else:
248     print(f"CONNECTION STATUS UNCLEAR")
249     print(f"    Issuer State: {issuer_state if issuer_state else 'N/A'}")
250     print(f"    Holder State: {holder_state if holder_state else 'N/A'}")
251     if conn_id_issuer_holder:
252         print(f"    Issuer Connection ID: {conn_id_issuer_holder}")
253     if conn_id_holder:
254         print(f"    Holder Connection ID: {conn_id_holder}")
255
256 print(f"{'='*60}")

```

Listing A-87*Jupyter Notebook Cell 9 Demonstration Kryptoagilität ed25519 Output*

```

1 Connection: Issuer --> Holder erstellen...
2
3 Pre-Check: Prüfe existierende Connections...
4
5 Issuer: 0 Connection(s) gefunden
6 Holder: 0 Connection(s) gefunden
7
8 Keine vollständige existierende Connection gefunden
9 Erstelle neue Connection mit did:peer:4...
10
11 Invitation-Modus: did:peer:4 (P2P Connection)
12 Issuer: Invitation erstellt
13 OOB ID: 6def3ce1-a8d2-49bd-ad8d-9a43a0fde93d
14 State: initial
15 Invitation Msg ID: a4d6d5e4-53b2-420b-a7c1-d50e3aa6efe9
16 Services: 1 service(s)
17 Label: Issuer Agent
18 Goal: Establish connection for credential issuance
19
20 Holder: Invitation akzeptiert
21 Connection ID (Holder): 539bd397-21c2-4330-9d61-428a0e5a544d
22
23 Issuer sucht Connection via invitation_msg_id...
24 Prüfe 1 Connection(s)...
25
26 Issuer: Connection gefunden via invitation_msg_id!
27     Connection ID: 3dd04340-dc11-4fc7-afda-86b011a9ac11
28     State: active
29     Invitation Msg ID: a4d6d5e4-53b2-420b-a7c1-d50e3aa6efe9
30     Their Label: Holder Agent
31     Updated At: 2025-12-12T12:03:58.769683Z
32
33 Post-Validation: Verifizierte Connection-Status...
34
35 Issuer Connection 3dd04340-dc11-4fc7-afda-86b011a9ac11:
36     State: active

```

```

37     Their Label: Holder Agent
38     Their Role: invitee
39     Invitation Msg ID: a4d6d5e4-53b2-420b-a7c1-d50e3aa6efe9
40
41     Holder Connection 539bd397-21c2-4330-9d61-428a0e5a544d:
42         State: active
43         Their Label: Issuer Agent
44         Their Role: inviter
45         Invitation Msg ID: a4d6d5e4-53b2-420b-a7c1-d50e3aa6efe9
46         Invitation Msg IDs stimmen überein!
47
48 =====
49 CONNECTION ESTABLISHED (did:peer:4): Issuer <--> Holder
50     Issuer Connection ID: 3dd04340-dc11-4fc7-afda-86b011a9ac11
51     Holder Connection ID: 539bd397-21c2-4330-9d61-428a0e5a544d
52     Issuer State: active
53     Holder State: active
54     Invitation Msg ID: a4d6d5e4-53b2-420b-a7c1-d50e3aa6efe9
55 =====

```

Listing A-88*Jupyter Notebook Cell 11 Demonstration Kryptoagilität ed25519*

```

1 # Cell 12 Wallet DID Übersicht anzeigen
2
3 print("Wallet-Übersicht - Alle Did's:\n")
4
5 # Alle Did's von allen Agenten abrufen
6 issuer_dids = api_get(ISSUER_ADMIN_URL, "/wallet/did")
7 holder_dids = api_get(HOLDER_ADMIN_URL, "/wallet/did")
8 verifier_dids = api_get(VERIFIER_ADMIN_URL, "/wallet/did")
9
10 #Zeige originale Responses
11 print("Originale /wallet/did Responses:\n")
12 if issuer_dids:
13     pretty_print(issuer_dids, "Issuer Wallet Did's")
14 if holder_dids:
15     pretty_print(holder_dids, "Holder Wallet Did's")
16 if verifier_dids:
17     pretty_print(verifier_dids, "Verifier Wallet Did's")

```

Listing A-89*Jupyter Notebook Cell 11 Demonstration Kryptoagilität ed25519 Output*

```

1 Wallet-Übersicht - Alle Did's:
2
3 Originale /wallet/did Responses:
4
5
6 =====
7     Issuer Wallet Did's
8 =====
9 {
10     "results": [
11         {

```

```

12     "did": "did:peer:4zQmVgo4trzrW244SytguXneycfvEqHsnsZKF7SibioFfk3a",
13     "verkey": "C4ypQhnn6R6g6Nth5LFrz7FNAwWjWiHSG1JDLfpRr3Ud",
14     "posture": "wallet_only",
15     "key_type": "ed25519",
16     "method": "did:peer:4",
17     "metadata": {}
18   },
19   {
20     "did": "did:peer:4zQmVgo4trzrW244SytguXneycfvEqHsnsZKF7SibioFfk3a:zKXbgJ6nwB...6
21         R7R5Tu6g8XDQSx",
22     "verkey": "C4ypQhnn6R6g6Nth5LFrz7FNAwWjWiHSG1JDLfpRr3Ud",
23     "posture": "wallet_only",
24     "key_type": "ed25519",
25     "method": "did:peer:4",
26     "metadata": {}
27   },
28   {
29     "did": "did:peer:4zQme8gBSG5x813d5KGKWig9tihua8dPSbENuqW7KNvkzolh:zKXbgJ...
30         gnfJBRT3cUDa",
31     "verkey": "GqqE1rZfqJ8koS7CYznRwm1729YxxKhyaAK9YeRcedKT",
32     "posture": "wallet_only",
33     "key_type": "ed25519",
34     "method": "did:peer:4",
35     "metadata": {
36       "invitation_reuse": "true"
37     }
38   }
39 ]
40 =====
41 Holder Wallet DIDs
42 =====
43 {
44   "results": [
45     {
46       "did": "did:peer:4zQmd6WegsSPVbYbxDG5gcTTPgV748PBeKg3fxVw3Q6fYysv",
47       "verkey": "9rVcKr3YMR3nCbcPcxyx8GyEhJ3rMtcwxuTtgoTpzpQv",
48       "posture": "wallet_only",
49       "key_type": "ed25519",
50       "method": "did:peer:4",
51       "metadata": {}
52     },
53     {
54       "did": "did:peer:4zQmd6WegsSPVbYbxDG5gcTTPgV748PBeKg3fxVw3Q6fYysv:zKXbgJ6nwBf...
55         XpiV5oNAvYMrxt",
56       "verkey": "9rVcKr3YMR3nCbcPcxyx8GyEhJ3rMtcwxuTtgoTpzpQv",
57       "posture": "wallet_only",
58       "key_type": "ed25519",
59       "method": "did:peer:4",
60       "metadata": {}
61     }
62   ]
63 }
64 =====
65 Verifier Wallet DIDs

```

```
66 =====
67 {
68     "results": []
69 }
```

Literaturverzeichnis

- Alam, M., Hoffstein, J., & Cambou, B. (2024). Privately Generated Key Pairs for Post Quantum Cryptography in a Distributed Network. *Applied Sciences* (2076-3417), 14(19), 8863. <https://doi.org/10.3390/app14198863>
- Alcaraz, C., & Zeadally, S. (2015). Critical infrastructure protection: Requirements and challenges for the 21st century. *International Journal of Critical Infrastructure Protection*, 8, 53–66. <https://doi.org/10.1016/j.ijcip.2014.12.002>
- Allende López, M. (2020, September). *Self-Sovereign Identity: The Future of Identity: Self-Sovereignty, Digital Wallets, and Blockchain*. Inter-American Development Bank. <https://doi.org/10.18235/0002635>
- Alsaqour, R., Majrashi, A., Alreedi, M., Alomar, K., & Abdelhaq, M. (2021). Defense in Depth: Multilayer of security. *International Journal of Communication Networks and Information Security (IJCNS)*, 13(2). <https://doi.org/10.17762/ijcns.v13i2.4951>
- Badertscher, C., Banfi, F., & Diaz, J. (2024). What Did Come Out of It? Analysis and Improvements of DIDComm Messaging. *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 4732–4746. <https://doi.org/10.1145/3658644.3690300>
- Bahce, A., & Utku, S. (2023). A Case Study for Mobile Wallet Implementation in Self-Sovereign Identity Infrastructure. *Journal of Artificial Intelligence and Data Science*, 3(1), 1–16. Verfügbar 2025-10-09 unter <https://dergipark.org.tr/tr/pub/jaida/issue/78404/1287040>
- Barrett-danes, F., & Ahmad, F. (2025). Quantum computing and cybersecurity: A rigorous systematic review of emerging threats, post-quantum solutions, and research directions (2019–2024). *Discover Applied Sciences*, 7(10), 1083. <https://doi.org/10.1007/s42452-025-07322-5>
- Berlato, S., Rizzi, M., Franzil, M., Cretti, S., De Matteis, P., & Carbone, R. (2024). Work-in-Progress: A Sidecar Proxy for Usable and Performance-Adaptable End-to-End Protection of Communications in Cloud Native Applications. *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 706–711. <https://doi.org/10.1109/EuroSPW61312.2024.00086>
- Bernstein, D. J. (2010). Grover vs. McEliece. In N. Sendrier (Hrsg.), *Post-Quantum Cryptography* (S. 73–80). Springer. https://doi.org/10.1007/978-3-642-12929-2_6
- Bernstein, D. J., Buchmann, J., Dahmén, E., & Bernstein, D. J. (2009). *Post-quantum cryptography*. Springer.
- Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., & Stebila, D. (2019). Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In J. Ding & R. Steinwandt (Hrsg.), *Post-Quantum Cryptography* (S. 206–226, Bd. 11505). Springer International Publishing. https://doi.org/10.1007/978-3-030-25510-7_12
- Boettiger, C. (2015). An Introduction to Docker for Reproducible Research. *SIGOPS Oper. Syst. Rev.*, 49(1), 71–79. <https://doi.org/10.1145/2723872.2723882>
- Bramer, W. M., De Jonge, G. B., Rethlefsen, M. L., Mast, F., & Kleijnen, J. (2018). A Systematic Approach to Searching: An Efficient and Complete Method to Develop Literature Searches. *Journal of the Medical Library Association*, 106(4). <https://doi.org/10.5195/jmla.2018.283>

- Brocke, J. vom, Simons, A., Riemer, K., Niehaves, B., Plattfaut, R., & Cleven, A. (2015). Standing on the Shoulders of Giants: Challenges and Recommendations of Literature Search in Information Systems Research. *Communications of the Association for Information Systems*, 37. <https://doi.org/10.17705/1CAIS.03709>
- BSI. (2022, 1. Dezember). *Empfehlungen zu Entwicklung und Einsatz von in Kritischen Infrastrukturen eingesetzten Produkten*. Bonn. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/KRITIS/UPK/upk-entwicklung-einsatz-produkte.pdf?__blob=publicationFile&v=13
- BSI. (2024, 10. September). *Konkretisierung Der KRITIS-Anforderungen (§ 8a Absatz 1 Und Absatz 1a BSIG)*. Bonn. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/KRITIS/Konkretisierung_Anforderungen_Massnahmen_KRITIS.pdf?__blob=publicationFile&v=20
- BSI. (2025a, 21. Januar). *Technische Richtlinie TR-02102-2 Kryptographische Verfahren: Empfehlungen Und Schlüssellängen Teil 2 – Verwendung von Transport Layer Security (TLS)* (Technische Richtlinie Nr. TR-02102-2). Bonn. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=11
- BSI. (2025b, 31. Januar). *BSI TR-02102-1 "Kryptographische Verfahren: Empfehlungen und Schlüssellängen" Version: 2025-01* (Technische Richtlinie). Bonn. Verfügbar 2025-11-19 unter https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI - TR - 02102 .pdf ? __blob = publicationFile&v=13
- Gesetz über das Bundesamt für Sicherheit in der Informationstechnik (BSI-Gesetz – BSIG) (2009, 14. August).
- Verordnung Zur Bestimmung Kritischer Infrastrukturen Nach Dem BSI-Gesetz (BSI-Kritisverordnung) (2016, 22. April). <https://www.gesetze-im-internet.de/bsi-kritisv/BSI-KritisV.pdf>
- Chhetri, G., Somvanshi, S., Hebli, P., Brotee, S., & Das, S. (2025, 12. Oktober). *Post-Quantum Cryptography and Quantum-Safe Security: A Comprehensive Survey*. <https://doi.org/10.48550/arXiv.2510.10436>
- Cyber Resilience Workshop Series Committee, Forum on Cyber Resilience, Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences & National Academies of Sciences, Engineering, and Medicine. (2017, 3. Februar). *Cryptographic Agility and Interoperability: Proceedings of a Workshop* (A. F. Johnson & L. I. Millett, Hrsg.). National Academies Press. <https://doi.org/10.17226/24636>
- Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung) (Text von Bedeutung für den EWR) (2016, 27. April). Verfügbar 2025-11-19 unter <http://data.europa.eu/eli/reg/2016/679/oj>
- De Iasio, A., & Zimeo, E. (2021). A framework for microservices synchronization. *Software: Practice and Experience*, 51(1), 25–45. <https://doi.org/10.1002/spe.2877>
- Demir, E. D., Bilgin, B., & Onbasli, M. C. (2025). Performance Analysis and Industry Deployment of Post-Quantum Cryptography Algorithms. <https://doi.org/10.48550/ARXIV.2503.12952>

- Dixit, A., Smith-Creasey, M., & Rajarajan, M. (2022). A Decentralized IIoT Identity Framework Based on Self-Sovereign Identity Using Blockchain. *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, 335–338. <https://doi.org/10.1109/LCN53696.2022.9843700>
- Feng, Z., Li, Z., Cui, H., & Whitty, M. T. (2025). Identity Management Systems: A Comprehensive Review. *Information*, 16(9), 778. <https://doi.org/10.3390/info16090778>
- Ferdous, M. S., Ionita, A., & Prinz, W. (2022). SSI4Web: A Self-sovereign Identity (SSI) Framework for the Web.
- Fraser, A., & Schneider, S. (2025). A Formal Security Analysis of Hyperledger AnonCreds. *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, 822–844. <https://doi.org/10.1109/EuroSP63326.2025.00052>
- Fritsch, L. (2020). Identity Management as a target in cyberwar. https://doi.org/10.18420/OIS2020_05
- Geremew, A., & Mohammad, A. (2024). Preparing Critical Infrastructure for Post-Quantum Cryptography: Strategies for Transitioning Ahead of Cryptanalytically Relevant Quantum Computing. *International Journal on Engineering, Science and Technology*, 6(4), 338–365. <https://doi.org/10.46328/ijonest.240>
- Ghosh, B. C., Ramakrishna, V., Govindarajan, C., Behl, D., Karunamoorthy, D., Abebe, E., & Chakraborty, S. (2021). Decentralized Cross-Network Identity Management for Blockchain Interoperation. *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 1–9. <https://doi.org/10.1109/ICBC51069.2021.9461064>
- Grover, L. K. (1996). A Fast Quantum Mechanical Algorithm for Database Search. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 212–219. <https://doi.org/10.1145/237814.237866>
- Hamza, H. S. (2005). Separation of concerns for evolving systems: A stability-driven approach. *Proceedings of the 2005 Workshop on Modeling and Analysis of Concerns in Software - MACS '05*, 1–5. <https://doi.org/10.1145/1083125.1083137>
- Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19.
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.*, 28(1), 75–105.
- ISO/IEC. (2022, Oktober). *ISO/IEC 27001:2022 Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements*.
- Jarkas, O., Ko, R., Dong, N., & Mahmud, R. (2025). A Container Security Survey: Exploits, Attacks, and Defenses. *ACM Computing Surveys*, 57(7), 1–36. <https://doi.org/10.1145/3715001>
- Jøsang, A. (2025). *Cybersecurity: Technology and Governance*. Springer Nature Switzerland. <https://doi.org/10.1007/978-3-031-68483-8>
- Khattar, K. (2025). Docker Like a Pro: Essential Practices for Secure and Scalable Containers. *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*, 09(02), 1–9. <https://doi.org/10.55041/IJSREM41900>
- Kreutzer, M., Wolf, L., & Waidner, M. (2024). Kryptoagilität. *Datenschutz und Datensicherheit - DuD*, 48(10), 667–672. <https://doi.org/10.1007/s11623-024-1997-8>

- Kuznetsov, O., Rusnak, A., Yezhov, A., Kuznetsova, K., Kanonik, D., & Domin, O. (2024). Merkle Trees in Blockchain: A Study of Collision Probability and Security Implications. *Internet of Things*, 26, 101193. <https://doi.org/10.1016/j.iot.2024.101193>
- Luo, S., Keller, M. S., Kakar, T., Choy, L., & Gehlenborg, N. (2025, 22. Oktober). *EasyVitessce: Auto-Magically Adding Interactivity to Scverse Single-Cell and Spatial Biology Plots*. <https://doi.org/10.48550/arXiv.2510.19532>
- Mamatha, G. S., Dimri, N., & Sinha, R. (2024, 18. März). *Post-Quantum Cryptography: Securing Digital Communication in the Quantum Era*. <https://doi.org/10.48550/arXiv.2403.11741>
- March, S. T., & Smith, G. F. (1995). Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4), 251–266. [https://doi.org/10.1016/0167-9236\(94\)00041-2](https://doi.org/10.1016/0167-9236(94)00041-2)
- Marchesi, L., Marchesi, M., & Tonelli, R. (2025). A Survey on Cryptoagility and Agile Practices in the Light of Quantum Resistance. *Information and Software Technology*, 178, 107604. <https://doi.org/10.1016/j.infsof.2024.107604>
- Martin, R. C. (2003). *Agile software development: Principles, patterns, and practices*. Prentice Hall/Pearson Education.
- Mazzocca, C., Acar, A., Uluagac, S., Montanari, R., Bellavista, P., & Conti, M. (2025). A Survey on Decentralized Identifiers and Verifiable Credentials. *IEEE Communications Surveys & Tutorials*, 27(6), 3641–3671. <https://doi.org/10.1109/COMST.2025.3543197>
- Meadows, C., Hounsinou, S., Wood, T., & Bloom, G. (2023). Sidecar-based Path-aware Security for Microservices. *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, 157–162. <https://doi.org/10.1145/3589608.3594742>
- Mehrez, H. A., & El Omri, O. (2018). The Crypto-Agility Properties. *Proceedings of The 12th International Multi-Conference on Society, Cybernetics and Informatics (IMSCI 2018)*. <https://www.iiis.org/cds2018/cd2018summer/papers/ha536vg.pdf>
- Moore, M., & Zenla, A. (2025, 8. Januar). *Goldilocks Isolation: High Performance VMs with Edera*. <https://doi.org/10.48550/arXiv.2501.04580>
- Moosavi, N., Taherdoost, H., Mohamed, N., Madanchian, M., Farhaoui, Y., & Khan, I. U. (2024). Blockchain Technology, Structure, and Applications: A Survey. *Procedia Computer Science*, 237, 645–658. <https://doi.org/10.1016/j.procs.2024.05.150>
- Myers, G. J., Sandler, C., & Badgett, T. (2012). *The art of software testing* (3rd ed). John Wiley & Sons. <https://doi.org/10.1002/9781119202486>
- Naghmouchi, M., & Laurent, M. (2025, 28. August). A Systematic Review and Layered Framework for Privacy-by-Design in Self-Sovereign Identity Systems. <https://doi.org/10.48550/arXiv.2502.02520>
- National Institute of Standards and Technology. (2024a, 13. August). *Module-Lattice-Based Digital Signature Standard (NIST FIPS 204)*. National Institute of Standards and Technology (U.S.) Washington, D.C. <https://doi.org/10.6028/NIST.FIPS.204>
- National Institute of Standards and Technology. (2024b, 13. August). *Module-Lattice-Based Key-Encapsulation Mechanism Standard (NIST FIPS 203)*. National Institute of Standards and Technology (U.S.) Washington, D.C. <https://doi.org/10.6028/NIST.FIPS.203>

- National Institute of Standards and Technology. (2024c, 13. August). *Stateless Hash-Based Digital Signature Standard (NIST FIPS 205)*. National Institute of Standards and Technology (U.S.) Washington, D.C. <https://doi.org/10.6028/NIST.FIPS.205>
- Nokhbeh Zaeem, R., Chang, K. C., Huang, T.-C., Liau, D., Song, W., Tyagi, A., Khalil, M., Lamison, M., Pandey, S., & Barber, K. S. (2021). Blockchain-Based Self-Sovereign Identity: Survey, Requirements, Use-Cases, and Comparative Study. *IEEE/WIC/ACM International Conference on Web Intelligence*, 128–135. <https://doi.org/10.1145/3486622.3493917>
- Nosouhi, M. R., Baig, Z., Doss, R., Gauravaram, P., Pati, D. P., Mahansaria, D., Sood, K., & Pan, L. (2024). The Value of Strong Identity and Access Management for ICS/OT Security. *2024 21st Annual International Conference on Privacy, Security and Trust (PST)*, 1–5. <https://doi.org/10.1109/PST62714.2024.10788047>
- Nouma, S. E., & Yavuz, A. A. (2024). Trustworthy and Efficient Digital Twins in Post-Quantum Era with Hybrid Hardware-Assisted Signatures. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 20(6), 1–30. <https://doi.org/10.1145/3638250>
- Osliaik, O., Saracino, A., Martinelli, F., & Mori, P. (2023). Cyber threat intelligence for critical infrastructure security. *Concurrency and Computation: Practice and Experience*, 35(23), e7759. <https://doi.org/10.1002/cpe.7759>
- Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., ... Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, n71. <https://doi.org/10.1136/bmj.n71>
- Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24, 45–77.
- Preukschat, A., & Reed, D. (2021). *Self-sovereign identity: Decentralized digital identity and verifiable credentials*. Manning.
- Radanliev, P. (2023). Review and Comparison of US, EU, and UK Regulations on Cyber Risk/Security of the Current Blockchain Technologies: Viewpoint from 2023. *The Review of Socionetwork Strategies*, 17(2), 105–129. <https://doi.org/10.1007/s12626-023-00139-x>
- Ramírez-Gordillo, T., Maciá-Lillo, A., Pujol, F. A., García-D'Urso, N., Azorín-López, J., & Mora, H. (2025). Decentralized Identity Management for Internet of Things (IoT) Devices Using IOTA Blockchain Technology. *Future Internet*, 17(1), 49. <https://doi.org/10.3390/fi17010049>
- Rescorla, E. (2018, August). *The Transport Layer Security (TLS) Protocol Version 1.3* (Request for Comments Nr. RFC 8446). Internet Engineering Task Force. <https://doi.org/10.17487/RFC8446>
- Rosa, G., Guglielmi, E., Iannone, M., Scalabrino, S., & Oliveto, R. (2025). Mining and measuring the impact of change patterns for improving the size and build time of docker images. *Empirical Software Engineering*, 30(5), 150. <https://doi.org/10.1007/s10664-025-10680-8>

- Rushby, J. (2004). Formal Methods and the Certification of Critical Systems. Verfügbar 2025-12-31 unter <https://www.semanticscholar.org/paper/Formal-Methods-and-the-Certification-of-Critical-Rushby-a34b9e54fd52b7243c22a1f35e95e045016ca920>
- Satybaldy, A., Ferdous, M. S., & Nowostawski, M. (2024). A Taxonomy of Challenges for Self-Sovereign Identity Systems. *IEEE Access*, 12, 16151–16177. <https://doi.org/10.1109/ACCESS.2024.3357940>
- Schardong, F., & Custódio, R. (2022). Self-Sovereign Identity: A Systematic Review, Mapping and Taxonomy. *Sensors*, 22(15), 5641. <https://doi.org/10.3390/s22155641>
- Sharif, A., Ranzi, M., Carbone, R., Sciarretta, G., Marino, F. A., & Ranise, S. (2022). The eIDAS Regulation: A Survey of Technological Trends for European Electronic Identity Schemes. *Applied Sciences*, 12(24), 12679. <https://doi.org/10.3390/app122412679>
- Shor, P. (1994). Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 124–134. <https://doi.org/10.1109/SFCS.1994.365700>
- Sikeridis, D., Kampanakis, P., & Devetsikiotis, M. (2020). Post-Quantum Authentication in TLS 1.3: A Performance Study. *Proceedings 2020 Network and Distributed System Security Symposium*. <https://doi.org/10.14722/ndss.2020.24203>
- Silver, M. S., Markus, M. L., & Beath, C. M. (1995). The Information Technology Interaction Model: A Foundation for the MBA Core Course. *MIS Quarterly*, 19(3), 361–390. <https://doi.org/10.2307/249600>
- Simon, H. A. (1996). *The sciences of the artificial* (3rd ed). MIT Press. OCLC: 47010985.
- Solavagione, A., & Vesco, A. (2025). Transition of Self-Sovereign Identity to Post-Quantum Cryptography. *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*, 174–181. <https://doi.org/10.1109/QCNC64685.2025.00035>
- Stebila, D., & Mosca, M. (2017). Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In R. Avanzi & H. Heys (Hrsg.), *Selected Areas in Cryptography – SAC 2016* (S. 14–37, Bd. 10532). Springer International Publishing. https://doi.org/10.1007/978-3-319-69453-5_2
- Strüker, J., Urbach, N., Guggenberger, T., Lautenschlager, J., Ruhland, N., Schlatt, V., Sedlmeir, J., Stoetzer, J.-C., & Völter, F. (2021). *Self-Sovereign Identity – Grundlagen, Anwendungen und Potenziale portabler digitaler Identitäten*. Fraunhofer-Institut für Angewandte Informationstechnik FIT, Projektgruppe Wirtschaftsinformatik. Bayreuth. https://www.fim-rc.de/wp-content/uploads/2021/06/Fraunhofer-FIT_SSI_Whitepaper.pdf
- Su, Y.-J., & Hsu, W.-C. (2023). Hyperledger Indy-based Roaming Identity Management System. *Taiwan Ubiquitous Information Journal*, 8(2), 545–557. <https://bit.kuas.edu.tw/~jni/2023/vol8/s2/16.JNI-0774.pdf>
- Swanzy, P. N., Abukari, A. M., & Ansong, E. D. (2024). Data Security Framework for Protecting Data in Transit and Data at Rest in the Cloud. *Current Journal of Applied Science and Technology*, 43(6), 61–77. <https://doi.org/10.9734/cjast/2024/v43i64387>

- Syahputra, R., & Arrozak, L. F. (2017). Power Transformer Loading Analysis in Order to Improve the Reliability of a Substation. *Journal of Electrical Technology UMY*, 1(4), 165–175. <https://doi.org/10.18196/jet.1422>
- Szymanski, T. H. (2024). A Quantum-Safe Software-Defined Deterministic Internet of Things (IoT) with Hardware-Enforced Cyber-Security for Critical Infrastructures. *Information*, 15(4), 173. <https://doi.org/10.3390/info15040173>
- Venable, J., Pries-Heje, J., & Baskerville, R. (2016). FEDS: A Framework for Evaluation in Design Science Research. *European Journal of Information Systems*, 25(1), 77–89. <https://doi.org/10.1057/ejis.2014.36>
- Zong, C. (2025). The Mathematical Foundation of Post-Quantum Cryptography. *Research (Washington, D.C.)*, 8, 0801. <https://doi.org/10.34133/research.0801>

Internetquellen

- About EBSCO Information Services.* (n. d.). Verfügbar 2025-05-30 unter <https://www.ebsco.com/about/mission>
- ACA-Py Docs.* (n. d.). Verfügbar 2025-12-19 unter <https://aca-py.org/latest/demo/AliceGetsAPhone/#run-an-instance-of-indy-tails-server>
- ACA-Py Plugins - ACA-Py Docs.* (n. d.). Verfügbar 2025-12-25 unter <https://aca-py.org/latest/features/PlugIns/>
- ACA-Py Plugins - ACA-Py Plugins.* (n. d.). Verfügbar 2025-12-25 unter <https://plugins.aca-py.org/latest/>
- Allen, C. (2016, 26. April). *The Path to Self-Sovereign Identity.* Life With Alacrity. Verfügbar 2025-12-12 unter <https://www.lifewithalacrity.com/article/the-path-to-self-sovereign-identity/>
- bcbgov. (n. d. a). *GitHub - bcbgov/indy-tails-server: This software stores and makes available tails files for use with Hyperledger Indy.* GitHub. Verfügbar 2025-11-26 unter <https://github.com/bcbgov/indy-tails-server>
- bcbgov. (n. d. b). *GitHub - bcbgov/von-network: A portable development level Indy Node network.* GitHub. Verfügbar 2025-11-26 unter <https://github.com/bcbgov/von-network>
- bcbgov. (n. d. c). *Indy-Tails-Server/Docker/Manage at Main · Bcbgov/Indy-Tails-Server · GitHub.* Verfügbar 2025-12-23 unter <https://github.com/bcbgov/indy-tails-server/blob/31e534ee0c4ece01a08e11ebd3d982b5be5d8b61/docker/manage>
- bcbgov. (n. d. d). *Indy-Tails-Server/README.Md at Main · Bcbgov/Indy-Tails-Server · GitHub.* Verfügbar 2025-12-22 unter <https://github.com/bcbgov/indy-tails-server/blob/31e534ee0c4ece01a08e11ebd3d982b5be5d8b61/README.md>
- bcbgov. (n. d. e). *Von-image node-1.12-6.* GitHub. Verfügbar 2025-12-20 unter <https://github.com/bcbgov/von-image/pkgs/container/von-image>
- bcbgov. (n. d. f). *Von-Network/Docker-Compose.Yml at Main · Bcbgov/von-Network · GitHub.* Verfügbar 2025-12-20 unter <https://github.com/bcbgov/von-Network/blob/f5f86aedb51d6dc295138ed7456e764591523fa0/docker-compose.yml>
- bcbgov. (n. d. g). *Von-Network/Docs/AddNewNode.Md at Main · Bcbgov/von-Network · GitHub.* Verfügbar 2025-12-20 unter <https://github.com/bcbgov/von-Network/blob/f5f86aedb51d6dc295138ed7456e764591523fa0/docs/AddNewNode.md>
- bcbgov. (n. d. h). *Von-Network/Docs/UsingVONNetwork.Md at Main · Bcbgov/von-Network · GitHub.* Verfügbar 2025-12-20 unter <https://github.com/bcbgov/von-Network/blob/f5f86aedb51d6dc295138ed7456e764591523fa0/docs/UsingVONNetwork.md>
- bcbgov. (n. d. i). *Von-Network/Scripts at Main · Bcbgov/von-Network · GitHub.* Verfügbar 2025-12-20 unter <https://github.com/bcbgov/von-Network/tree/f5f86aedb51d6dc295138ed7456e764591523fa0/scripts>
- Building Nginx from Sources.* (n. d.). Verfügbar 2025-12-19 unter <https://nginx.org/en/docs/configure.html>
- Canonical Ltd. (2024). *Ubuntu Server 24.04 LTS Documentation.* https://documentation.ubuntu.com/_downloads/server/en/latest/pdf/
- Chromium Docs - Checking out and Building Chromium on Linux.* (n. d.). Verfügbar 2025-11-28 unter https://chromium.googlesource.com/chromium/src/+/main/docs/linux/build_instructions.md

- decentralized-identity. (n. d. a). *Aries-rfcs/features/0023-did-exchange at main · decentralized-identity/aries-rfcs*. GitHub. Verfügbar 2025-12-23 unter <https://github.com/decentralized-identity/aries-rfcs/tree/56dad506841420f2407af858b9c4ea69013ada19/features/0023-did-exchange>
- decentralized-identity. (n. d. b). *Aries-rfcs/features/0434-outofband/README.md at main · decentralized-identity/aries-rfcs*. GitHub. Verfügbar 2025-12-23 unter <https://github.com/decentralized-identity/aries-rfcs/blob/56dad506841420f2407af858b9c4ea69013ada19/features/0434-outofband/README.md>
- decentralized-identity. (n. d. c). *Aries-rfcs/features/0453-issue-credential-v2 at main · decentralized-identity/aries-rfcs*. GitHub. Verfügbar 2025-12-23 unter <https://github.com/decentralized-identity/aries-rfcs/tree/56dad506841420f2407af858b9c4ea69013ada19/features/0453-issue-credential-v2>
- decentralized-identity. (n. d. d). *Did-Peer-4/README.Md at Main · Decentralized-Identity/Did-Peer-4*. GitHub. Verfügbar 2026-01-06 unter <https://github.com/decentralized-identity/did-peer-4/blob/1c95404a7b6787001a86ce7e5a36254c9688ac0f/README.md>
- Docker Inc. (n. d. a). *Docker Compose*. Docker Documentation. Verfügbar 2026-01-04 unter <https://docs.docker.com/compose/>
- Docker Inc. (n. d. b). *What is Docker?* Docker Documentation. Verfügbar 2026-01-04 unter <https://docs.docker.com/get-started/docker-overview/>
- Entry in Aries_askar::Entry - Rust.* (n. d.). Verfügbar 2025-12-26 unter https://docs.rs/aries-askar/latest/aries_askar/entry/struct.Entry.html
- Github. (n. d.). *About Git*. GitHub Docs. Verfügbar 2026-01-04 unter <https://docs-internal.github.com/en/get-started/using-git/about-git>
- Hardman, D. (2018, 1. Februar). *0011: Credential Revocation — Hyperledger Indy HIPE Documentation*. Verfügbar 2025-11-23 unter <https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README/>
- hyperledger. (n. d.). *Indy Node Troubleshooting Guide — Hyperledger Indy Node Documentation*. Verfügbar 2025-12-20 unter <https://hyperledger-indy.readthedocs.io/projects/node/en/latest/troubleshooting/>
- Microsoft. (n. d.). *Windows 10 Home und Pro - Microsoft Lifecycle*. Verfügbar 2026-01-04 unter <https://learn.microsoft.com/de-de/lifecycle/products/windows-10-home-and-pro>
- Microsoft. (2025, 13. August). *Hyper-V Virtualisierung in Windows Server und Windows*. Verfügbar 2026-01-04 unter <https://learn.microsoft.com/de-de/windows-server/virtualization/hyper-v/overview>
- multiformats. (2025, 11. Dezember). *Multiformats/Multicodec*. Verfügbar 2025-12-26 unter <https://github.com/multiformats/multicodec>
- nginx. (n. d.). *GitHub - nginx/nginx: The official NGINX Open Source repository*. GitHub. Verfügbar 2025-11-26 unter <https://github.com/nginx/nginx>
- open-quantum-safe. (n. d. a). *Oqs-demos/chromium at 643ef99297fe8c6ebd3587b5dd238d5e7a457037 · open-quantum-safe/oqs-demos*. GitHub. Verfügbar 2025-11-28 unter <https://github.com/open-quantum-safe/oqs-demos/tree/643ef99297fe8c6ebd3587b5dd238d5e7a457037/chromium>

- open-quantum-safe. (n. d. b). *Oqs-demos/chromium/README-Linux.md* at 643ef99297fe8c6ebd3587b5dd238d5e7a457037 · open-quantum-safe/oqs-demos. GitHub. Verfügbar 2025-11-28 unter <https://github.com/open-quantum-safe/oqs-demos/blob/643ef99297fe8c6ebd3587b5dd238d5e7a457037/chromium/README-Linux.md>
- open-quantum-safe. (2025, 3. November). *Open-Quantum-Safe/Oqs-Demos Nginx Dockerfile*. Verfügbar 2025-11-24 unter <https://raw.githubusercontent.com/open-quantum-safe/oqs-demos/refs/heads/main/nginx/Dockerfile>
- openwallet-foundation. (n. d. a). *Acapy/Acapy_agent/Admin/Server.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/admin/server.py
- openwallet-foundation. (n. d. b). *Acapy/Acapy_agent/Askar/Didcomm/v1.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/askar/didcomm/v1.py
- openwallet-foundation. (n. d. c). *Acapy/Acapy_agent/Askar/Store.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/askar/store.py
- openwallet-foundation. (n. d. d). *Acapy/Acapy_agent/Protocols/Didexchange/V1_0/Manager.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/protocols/didexchange/v1_0/manager.py
- openwallet-foundation. (n. d. e). *Acapy/Acapy_agent/Protocols/Issue_credential/V2_0/Manager.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/protocols/issue_credential/v2_0/manager.py
- openwallet-foundation. (n. d. f). *Acapy/Acapy_agent/Protocols/Issue_credential/V2_0/Routes.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/protocols/issue_credential/v2_0/routes.py
- openwallet-foundation. (n. d. g). *Acapy/Acapy_agent/Protocols/Out_of_band/V1_0/Manager.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/protocols/out_of_band/v1_0/manager.py
- openwallet-foundation. (n. d. h). *Acapy/Acapy_agent/Protocols/Out_of_band/V1_0/Routes.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/protocols/out_of_band/v1_0/routes.py

- openwallet-foundation. (n. d. i). *Acapy/Acapy_agent/Transport/Inbound/Http.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/transport/inbound/http.py
- openwallet-foundation. (n. d. j). *Acapy/Acapy_agent/Transport/Outbound/Http.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/transport/outbound/http.py
- openwallet-foundation. (n. d. k). *Acapy/Acapy_agent/Wallet/Askar.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/wallet/askar.py
- openwallet-foundation. (n. d. l). *Acapy/Acapy_agent/Wallet/Key_type.py* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/acapy_agent/wallet/key_type.py
- openwallet-foundation. (n. d. m). *Acapy/README.Md* at Main · Openwallet-Foundation/Acapy · GitHub. Verfügbar 2025-12-23 unter <https://github.com/openwallet-foundation/acapy/blob/d7527214a44103325ff5a0331f9a4003d37edeba/README.md>
- openwallet-foundation. (n. d. n). *GitHub - openwallet-foundation/acapy: ACA-Py is a foundation for building decentralized identity applications and services running in non-mobile environments.* GitHub. Verfügbar 2025-11-26 unter <https://github.com/openwallet-foundation/acapy>
- openwallet-foundation. (2025, 23. Dezember). *Openwallet-Foundation/Acapy-Plugins*. Verfügbar 2025-12-25 unter <https://github.com/openwallet-foundation/acapy-plugins>
- Poetry - Python Dependency Management and Packaging Made Easy.* (n. d.). Verfügbar 2025-12-28 unter <https://python-poetry.org/>
- Turner, A. (n. d.). *What's New In Python 3.12.* Python documentation. Verfügbar 2026-01-04 unter <https://docs.python.org/3/whatsnew/3.12.html>
- w3c. (2025, 15. Mai). *Controlled Identifiers v1.1.* Verfügbar 2025-12-26 unter <https://w3c.github.io/cid/#choosing-a-multiformat>

KI-Hilfsmittelverzeichnis

Die in Tabelle A-12 gelisteten Generative KI-Tools wurden in einem iterativen Prozess als kritische Analyseinstrumente in interaktiver, dialogischer Weise eingesetzt. Ausgehend von selbst verfassten Textentwürfen und literaturgestützten Kernaussagen wurden mehrere Optimierungszyklen durchlaufen, in denen die KI zur Schärfung von Formulierungen, Präzisierung der Argumentationslogik und Verbesserung der sprachlichen Kohärenz beitrug. Die KI fungierte dabei als Qualitätssicherungsinstanz für bereits konzipierte Inhalte, nicht als Inhaltsgenerator. Im technischen Teil unterstützte die KI durch Code-Validierung, der Identifikation logischer Fehler, Effizienzoptimierung und Syntaxkonformitätsprüfung während der iterativen PQC-SSI-Prototyp-Entwicklung.

Tabelle A-12
KI-Hilfsmittelverzeichnis

KI-Tool	Beschreibung	Einsatzbereich
Claude	Vielseitiges Sprachmodell	Optimierung und
4.5 Sonnet	von Anthropic mit erweiterten Reasoning- und Computer-Use-Fähigkeiten.	Syntaxkontrolle des selbst verfassten Source-Codes und LaTeX-Korrektur
Perplexity.ai	Reasoning-Modell mit integrierter Web-Suche und Chain-of-Thought-Fähigkeiten.	Recherche, Textvalidierung, Faktencheck
Sonar Reasoning		
Github Copilot 1.0	KI-gestützte Programmierhilfe zur Analyse und Optimierung der LaTeX-Syntax	Textvalidierung, Rechtschreib- und Grammatik-Prüfung innerhalb der LaTeX-Dokumente

Anmerkung. Eigene Darstellung der eingesetzten KI-Hilfsmittel.

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

München, 6.1.2026

(Ort, Datum)



Ferris Florian Menzel