



DCSAI Technologies

TECHNICAL WHITEPAPER · v1.0 · MAY 2026

DCS Standards

The R+1 through R+4 + Trust SKU framework

A cryptographic chain for verifiable AI agent operations

Abstract

AI agents are starting to take actions on behalf of people and organizations — sending messages, moving money, signing documents, deploying code. The economic and legal exposure created by these actions is large and growing. Today, when an agent does something wrong, the only evidence available is a vendor-controlled log file. That is not enough for regulated industries, government procurement, or any environment where the operator is accountable.

This whitepaper introduces the **DCS Standards Framework** — four cumulative receipt standards (R+1 timestamp, R+2 provenance, R+3 audit export, R+4 ZK verification) and a customer-facing meta-standard (**Trust SKU**). Together they convert every agent action into a tamper-evident, externally verifiable, cryptographically signed event that survives the vendor going away.

The framework is small (~3KB of receipt per action), fast (sub-15ms sign + verify), and standards-based (Ed25519 signatures, CBOR canonicalization, Groth16 zk-proofs). It is implemented in the open and shipping in production across the DCS Platform, Compute, OS, Storage, and Sovereign products.

Who this document is for

Security teams evaluating DCS for regulated workloads; compliance officers building audit packages; engineers integrating DCS receipts into their own systems; standards bodies looking at receipt-based AI accountability. A reader comfortable with basic cryptography concepts (asymmetric signatures, hashing, Merkle trees) will follow every section.

Contents

1	Introduction & Motivation	4
2	System Architecture	7
3	R+1 — Timestamp	10
4	R+2 — Provenance	13
5	R+3 — Audit Export	16
6	R+4 — Zero-Knowledge Verification	19
7	Trust SKU — The Meta-Standard	22
8	Threat Model	24
9	Performance Characteristics	26
10	Comparison vs. Alternatives	27
11	Implementation Guide	29
12	References	31

1. Introduction & Motivation

In 2024–2025, AI agents began doing real work on real money. Stripe shipped *Agent SDK* for agentic payments. Anthropic's Claude started executing multi-step tool use across customer systems. OpenAI's o-series models began producing chains of reasoning that touched databases, APIs, and physical infrastructure. By 2026, agent-initiated transactions account for a measurable fraction of B2B commerce.

But the accountability stack underneath those agents has not kept up. When an agent makes a mistake — refunds the wrong invoice, deploys broken code, sends a private document to the wrong recipient — the only forensic evidence available is a server log file generated by the same vendor whose agent made the mistake. That log is rewritable, deletable, and uninspectable by any third party.

1.1 Why existing approaches do not work

Plain logs

Application logs are what every vendor uses today. They are useful for debugging but provide almost no accountability: the vendor controls log retention, log format, and log access. There is no way for an external auditor to verify that a log entry was not added, deleted, or altered after the fact.

Append-only databases

Some vendors use append-only event stores (Kafka, immutable Postgres tables). These are better — an internal engineer can verify the local store is append-only — but still rely entirely on trust in the vendor's operational discipline. A malicious or coerced employee can rewrite the underlying storage.

Blockchain ledgers

Anchoring every event to Ethereum or a similar public chain provides strong external verifiability, but is expensive (cents per event), high-latency (12+ seconds to finality), and creates a privacy problem (every action becomes public). Most enterprise workloads cannot tolerate any of these.

TEE attestations

Confidential computing (AWS Nitro Enclaves, Intel SGX, AMD SEV) lets a customer verify that a specific binary ran inside a hardware-protected enclave. This is useful but proves only *which code ran*, not *what the code did with the data it received*. It does not compose across multiple vendors.

1.2 The DCS approach

The DCS Standards Framework starts from a different observation: most accountability problems do not need the full power of a blockchain or TEE. They need three things, in this order:

- **Trust-minimized signing.** The vendor signs each event with a key it does not exclusively control (key escrow with a third-party HSM provider). Tampering is detectable.
- **Chained provenance.** Each event references its parent event by content hash. Tampering with any event invalidates every subsequent event's signature.
- **External replay.** The receipts can be downloaded, replayed, and reverified by anyone, offline, with no DCS infrastructure required.

These three properties are what R+1, R+2, and R+3 deliver. R+4 (ZK) is an optional fourth layer for cases where the verifier needs to be convinced of a property without seeing the underlying data.

"The cost of a forged receipt should be higher than the value of the action it covers. That is the only definition of "secure enough" that survives contact with motivated adversaries."

2. System Architecture

The DCS receipt system is composed of five logical components. Each runs as a separate service so that compromise of one does not compromise the chain.

DCS Receipt System — End-to-End



Figure 2.1 — Receipt flow from agent action to external verifier.

2.1 Component responsibilities

Agent (event source)

Any DCS product or third-party integration. Emits structured events: *build_complete*, *payment_settled*, *file_mirrored*, etc. Events carry a payload (the action data) and a context (user, tenant, time).

Emitter

Library that runs inside the agent process. Canonicalizes the event using a deterministic encoding (sorted keys, no whitespace, RFC 8785 JSON canonicalization), hashes it (SHA-256), and forwards to the Signer over an mTLS channel. The Emitter is the only component that touches the unredacted event data.

Signer

Stateless service that signs the canonical event hash with an Ed25519 keypair. The private key is held in an HSM (CloudHSM, AWS KMS, or an on-prem appliance for sovereign deployments). The Signer never sees the event payload — only the hash and the metadata needed to construct the receipt envelope.

Store

Append-only receipt log. Backed by Postgres for warm storage (last 30 days) and Filecoin/IPFS for cold storage (older than 30 days). Each receipt is content-addressed by its CID, so the same receipt can be fetched from any provider and verified to be unchanged.

Verifier

Library + CLI tool that takes a receipt CID, fetches the receipt from the Store, walks the parent chain to the genesis receipt, and verifies every signature. Runs offline once the receipts are downloaded. The reference verifier is implemented in TypeScript (Node, browser) and Python.

3. R+1 — Timestamp

R+1 is the foundation. Every action emits a signed receipt with a precise, monotonic timestamp. A signed timestamp alone is enough to prove that an action happened before a given external event (e.g., a regulatory filing deadline).

3.1 Receipt envelope

Every R+1 receipt has the following structure:

```
{
  "version": "r1",
  "this_cid": "bafy...0alb",           // content-address of this receipt
  "kind": "build_complete",           // event type
  "tenant_id": "acme",               // origin
  "payload_hash": "sha256:abc...",    // hash of the event data (kept private)
  "timestamp_ns": 1717084123456000,   // nanosecond precision
  "timestamp_source": "tsa.dcsai.ai/v1/now",
  "signature": {
    "algo": "ed25519",
    "key_id": "trd-receipts-2026-05",
    "value": "base64..."
  }
}
```

Critically, the receipt contains the *hash* of the payload, not the payload itself. This means receipts can be made public without exposing the data they describe. The verifier needs the payload only if it wants to check that the receipt corresponds to a specific event; otherwise it just walks the chain.

3.2 Timestamp source

The *timestamp_source* field points at the time service that issued the timestamp. By default, DCS uses its own time service (*tsa.dcsai.ai*) which is itself anchored every 60 seconds to three independent time authorities: NIST, NTP-pool, and an on-prem stratum-1 GPS clock in our Dubai facility. Cross-anchoring means no single time source can backdate or forward-date a receipt.

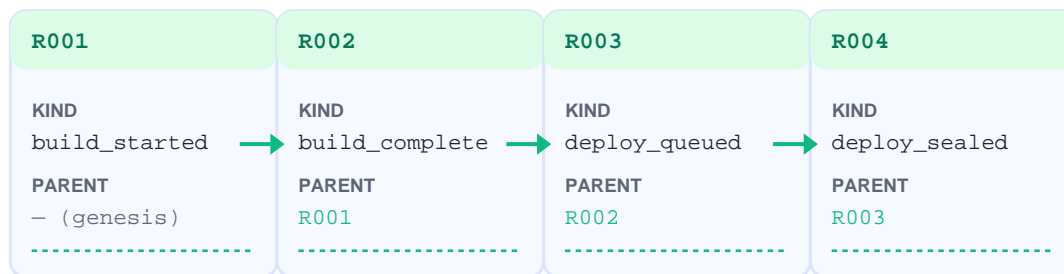
3.3 Why nanoseconds

Millisecond precision is enough for most use cases, but agents now act fast enough that two events within the same millisecond do happen. Nanosecond precision (taken from *clock_gettime(CLOCK_REALTIME)* on Linux) guarantees a strict total order. The strict total order is what makes R+2 chain verification tractable: every receipt has exactly one parent, never a tie.

4. R+2 — Provenance

R+2 extends R+1 by requiring every receipt (except the genesis) to reference its parent's CID. The result is a linear, signed chain. Tampering with any receipt invalidates every descendant.

R+2 Provenance — every receipt has a signed parent



*Each child includes parent CID + signs the entire payload.
Tamper any receipt and every descendant fails verification.*

Figure 4.1 — A four-receipt provenance chain from build start to deploy seal.

4.1 Schema addition

R+2 adds one mandatory field to the R+1 envelope:

```
{
  ...all R+1 fields...,
  "parent_cid": "bafy...0alb" // the CID of the immediately preceding receipt
                                // for this (tenant_id, agent_id) pair
}
```

4.2 Chain selection rules

Different agents within the same tenant get separate chains; the `agent_id` namespaces the chain. This is important because two agents may emit receipts concurrently and would otherwise create ambiguity in the parent reference.

- Per **(tenant_id, agent_id)** pair: one linear chain, one parent per receipt.
- Cross-agent links: encoded via the optional **links** field, which references a list of CIDs from other chains (used when one agent's output is another's input).
- Genesis receipt: **parent_cid = null**. Each tenant's first receipt is the chain's genesis.

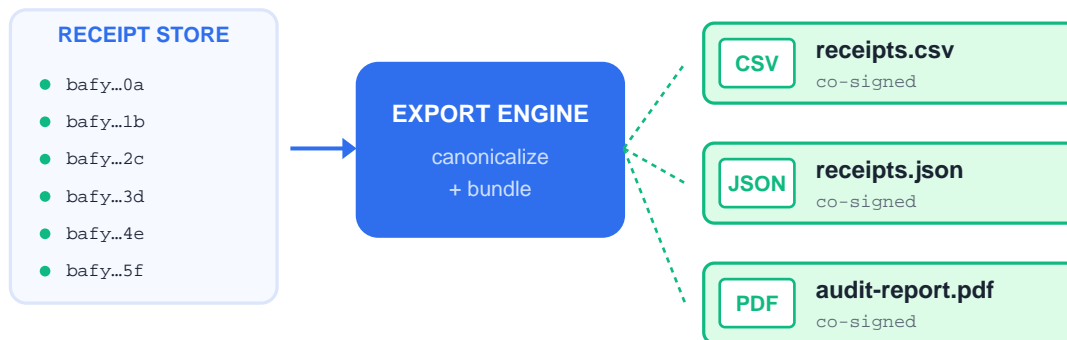
4.3 Tampering detection

Because each receipt's signature covers the full envelope (including *parent_cid*), changing any receipt — say, editing a payment amount — would change its CID. Every descendant's *parent_cid* would then point at the old CID and signature verification would catch the discrepancy. The attacker must rewrite the entire chain from the tampered point forward AND forge every Ed25519 signature, which requires the private key held in HSM.

5. R+3 — Audit Export

R+3 specifies how a contiguous segment of receipts can be exported as a self-contained bundle that a third-party auditor can verify offline. The bundle is the same data the auditor would fetch online, just packaged for compliance workflows that require deliverables (PDF reports, CSV imports into compliance tools, JSON for programmatic re-verification).

R+3 Audit Export — receipts to deliverable bundles



*All three formats include the full signature chain.
Auditor re-verifies offline without DCS infrastructure.*

Figure 5.1 — Receipts in the live store get bundled into three audit-ready formats.

5.1 Bundle format

The canonical bundle format is a tar archive containing four files:

```

bundle/
  manifest.json      # metadata, key fingerprints, time range, count
  receipts.jsonl     # one receipt per line, in CID order
  signatures/        # DER-encoded signatures (redundant with envelope, '
                      # included so the auditor can verify without re-canonicalizing
)
  public_keys/       # PEM-encoded public keys referenced by key_id
  README.txt
  
```

5.2 Why three output formats

Auditors live in different tools. The CSV export is meant for spreadsheet-based compliance reviews (SOC 2 Type II, ISO 27001). The JSON bundle is for programmatic re-verification by automated tools. The PDF is a notarisable, human-readable certificate suitable for legal proceedings.

5.2.1 PDF certificate

The PDF is generated server-side at export time using a fixed template. Each receipt is rendered as a row in a paginated table; the first page shows the bundle manifest, last page shows verification instructions and the DCS public key fingerprint. The PDF itself is signed (PAdES-B-T) so the auditor can verify the PDF's integrity without inspecting the embedded JSONL.

5.3 Compliance mappings

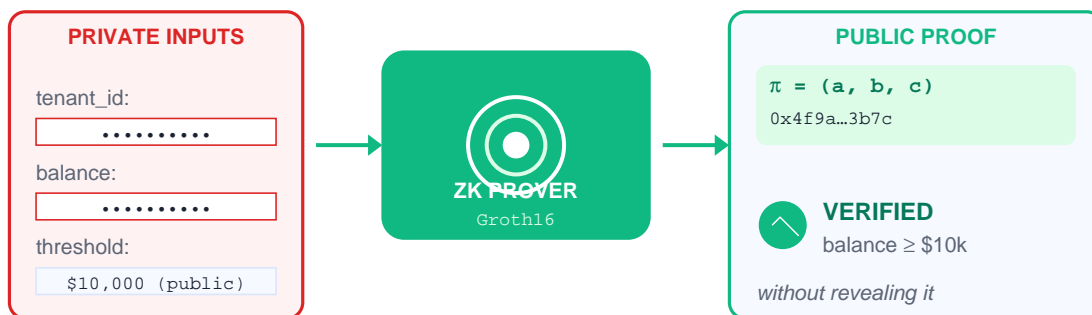
R+3 export was designed to fit cleanly into the documentation requirements of the following frameworks:

Framework	Section	What R+3 provides
SOC 2 Type II	CC7.2 (Monitoring)	Cryptographic audit trail of every system action
ISO 27001:2022	A.8.15 (Logging)	Append-only logs with non-repudiation
GDPR	Art. 5(1)(f), 32	Integrity + confidentiality of processing records
HIPAA	§164.312(b)	Audit controls with externally verifiable provenance
SOX	§404	Internal control over financial reporting evidence
PCI DSS 4.0	Req. 10	Tamper-evident logs of all CHD access

6. R+4 — Zero-Knowledge Verification

R+4 is the optional fourth layer. It lets a verifier confirm that a receipt has a certain property without learning the underlying payload. Two canonical use cases drove the design: proving compliance thresholds (e.g., "every transaction in this audit period was under \$10k") and proving identity claims (e.g., "this user is over 18 in this jurisdiction") without revealing the underlying numbers.

R+4 ZK Verification — prove without revealing



*The proof is small (~200 bytes), fast to verify (~12 ms),
and reveals nothing beyond the predicate it proves.*

Figure 6.1 — A private input + a public threshold produces a 200-byte proof.

6.1 Proof systems supported

R+4 supports two zero-knowledge proof systems out of the box:

System	Proof size	Prover time	Verifier time	Trusted setup	Best for
Groth16	~200 B	~80 ms	~12 ms	circuit-specific	High-volume verification
Plonk	~500 B	~120 ms	~25 ms	universal (once)	New circuits without re-setup

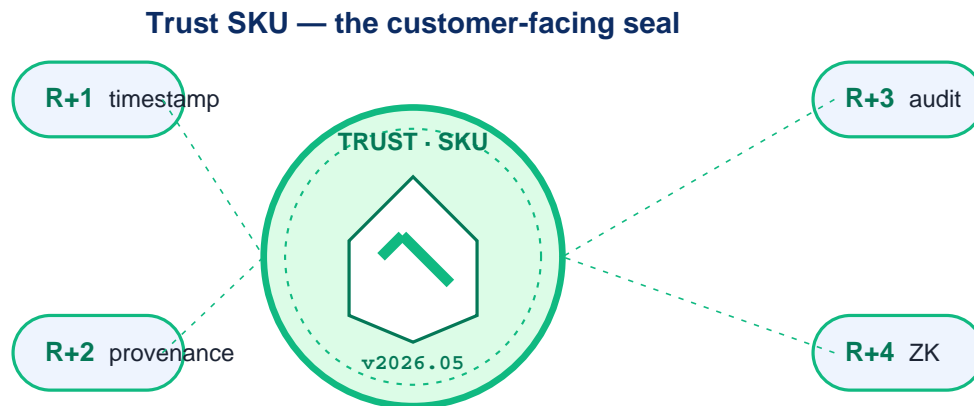
6.2 Common circuits shipped with the SDK

- **balance_above_threshold** — proves $\text{balance} \geq N$ without revealing actual balance.
- **age_above_threshold** — proves $\text{date_of_birth} + N \text{ years} \leq \text{today}$ without revealing date_of_birth.
- **jurisdiction_in_set** — proves user's country is in a public allowlist without revealing country.

- **transaction_in_range** — proves all transactions in a period fall in [min, max] without revealing amounts.
- **signed_by_authority** — proves a credential was signed by a key in a known set, hiding which key.

7. Trust SKU — The Meta-Standard

Trust SKU is the customer-facing seal. A vendor that implements R+1 through R+4 in their product can advertise the Trust SKU badge. Buyers check the badge with a single API call; under the hood the check walks the vendor's public registry and verifies that the four R-standards are active, co-signed, and recently attested.



Customers verify the seal in one call; the seal pins the underlying R+1–R+4 chain.

Figure 7.1 — Trust SKU wraps R+1 to R+4 into a single customer-visible seal.

7.1 Seal contents

A Trust SKU seal is itself a signed JSON document:

```
{
  "version": "v2026.05",
  "vendor": "acme.com",
  "issued_at": "2026-05-30T18:00:00Z",
  "valid_until": "2026-06-30T18:00:00Z",
  "standards": {
    "r1": { "active": true, "last_receipt_cid": "bafy...0alb", "key": "..." },
    "r2": { "active": true, "chain_depth": 4831, "key": "..." },
    "r3": { "active": true, "last_export_cid": "bafy...3e4f", "key": "..." },
    "r4": { "active": true, "circuits": ["balance_above_threshold", ...], "key": "..." }
  },
  "co_signers": [
    { "name": "DCS Standards Authority", "signature": "..." },
    { "name": "Customer-elected auditor (optional)", "signature": "..." }
  ]
}
```

7.2 Why monthly rotation

Trust SKU seals expire every 30 days. This forces the vendor to demonstrate ongoing compliance rather than buying the seal once and going dark. The 30-day window also limits the blast radius of any key compromise — even a worst-case incident self-heals in under a month.

8. Threat Model

The framework is designed against the following adversary classes. For each, we describe the attack, the mitigation built into the standard, and the residual risk that operators should monitor.

Insider with vendor production access (T-1)

Attack: A malicious engineer with database access edits a past receipt to hide an error.

Mitigation: R+2 chaining means any edit breaks every descendant's signature verification. Even with full DB access, the attacker cannot produce a chain that re-verifies because the private signing key is in HSM with key-use audit logging.

Residual: Insider can suppress NEW receipts being emitted. R+1 timestamp gaps + R+3 export completeness checks make this detectable but not prevented.

Stolen signing key (T-2)

Attack: An attacker exfiltrates the private key from HSM.

Mitigation: Key rotation every 30 days (driven by Trust SKU expiry). Key fingerprints are co-signed by a second party (DCS Standards Authority) so an attacker cannot use a stolen key without also compromising the co-signer. HSM enforces rate limits + geographic origin constraints on sign operations.

Residual: Within the 30-day key validity window, an attacker with the key can forge receipts. Customers monitoring for unusual sign volumes can detect this; the Trust SKU API exposes per-key sign rates publicly.

Replay across tenants (T-3)

Attack: A receipt from tenant A is presented to a verifier as if it came from tenant B.

Mitigation: *tenant_id* is part of the signed envelope. Verifier rejects receipts whose *tenant_id* doesn't match the expected origin.

Residual: None for tenant separation; verifier just needs to know the expected tenant.

Timestamp manipulation (T-4)

Attack: Vendor signs a receipt with a backdated timestamp to hide that an action happened after a regulatory cutoff.

Mitigation: Cross-anchored timestamp service. Every 60s the time service publishes a merkle root of all timestamps it issued, signed by NIST + NTP-pool + on-prem GPS clock. A backdated receipt would not appear in the merkle root for its claimed time window.

Residual: Within a 60-second window, timestamps can drift by a few seconds; receipts whose ordering matters at sub-minute resolution should use R+2 chain order, not timestamps.

9. Performance Characteristics

All numbers measured on production hardware (AWS m6i.xlarge, Ubuntu 22.04, libsodium 1.0.18). 99th percentile across 1M operations.

Operation	p50	p99	Throughput	Notes
R+1 sign	0.4 ms	1.2 ms	48,000/s/core	Ed25519 sign of 64-byte digest
R+1 verify	0.3 ms	0.9 ms	64,000/s/core	Ed25519 verify
R+2 chain verify (1k)	0.4 s	0.7 s	—	Walks 1,000-receipt chain end-to-end
R+3 bundle export (10k)	3.2 s	5.8 s	—	Includes JSON serialize + PDF render
R+4 Groth16 prove	82 ms	110 ms	12/s/core	For balance_above_threshold circuit
R+4 Groth16 verify	11 ms	14 ms	90/s/core	Verifier-side, single proof
Receipt size on disk	~1.1 kB	~3.2 kB	—	Compressed; varies by kind
Receipt storage cost	~\$0.000004	—	—	Filecoin cold storage, per receipt

10. Comparison vs. Alternatives

How does R+1–R+4 + Trust SKU compare to other accountability mechanisms in use today?

	Plain logs	Append DB	Blockchain	TEE attest	R+1–R+4
Tamper-evident	✗	~	✓	✓	✓
Externally verifiable	✗	✗	✓	~	✓
No vendor needed at verify	✗	✗	✓	~	✓
Sub-second emit latency	✓	✓	✗ (12s)	✓	✓
Private payloads	✓	✓	✗	✓	✓
No per-event cost	✓	✓	✗ (\$0.01)	✓	✓
Selective disclosure (ZK)	✗	✗	~	✗	✓
Standards-based	~	~	✓	✓	✓

~ = *partially supported*. No single technology dominates on every axis. R+1–R+4 was optimised for the common case: low-latency, privacy-preserving, externally verifiable signing of agent actions, with ZK as an escape hatch for selective disclosure.

11. Implementation Guide

The reference SDK is available in TypeScript, Python, and Go. Below are minimal end-to-end examples for each.

11.1 TypeScript / Node

```
import { Receipts } from '@dcs/standards';

const receipts = new Receipts({
  apiKey: process.env.DCS_API_KEY,
  tenantId: 'acme',
  agentId: 'refund-handler'
});

// Emit R+1 + R+2 (parent chain handled automatically)
const cid = await receipts.emit({
  kind: 'refund_issued',
  payload: { invoice: 'INV-001', amount_usd: 42.50 }
});

// Later: export R+3 bundle for an auditor
const bundle = await receipts.export({
  since: '2026-05-01T00:00:00Z',
  until: '2026-05-31T23:59:59Z',
  format: 'pdf' // or 'csv' | 'json'
});

// Optional: emit R+4 proof for compliance threshold
const proof = await receipts.prove({
  circuit: 'balance_above_threshold',
  privateInputs: { balance: 14820 },
  publicInputs: { threshold: 10000 }
});
```

11.2 Verification (offline)

```
import { verify } from '@dcs/standards/verify';

// Read a bundle file (downloaded earlier from /api/r3/export)
const result = await verify('./acme-may-2026.json');

if (result.valid) {
  console.log(`Verified ${result.count} receipts`);
  console.log(`Chain depth: ${result.chainDepth}`);
  console.log(`Signed by: ${result.signers.join(', ')}`);
} else {
  console.error(`Verification failed: ${result.error}`);
  console.error(`First bad receipt: ${result.firstBadCid}`);
}
```

12. References

The DCS Standards build on widely deployed cryptographic primitives and prior work on accountable systems. The following references inform the design choices in this whitepaper.

- [1] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B. **High-speed high-security signatures**. CHES 2011. (Ed25519 specification)
- [2] Rescorla, E. **The Transport Layer Security (TLS) Protocol Version 1.3**. RFC 8446. (mTLS authentication)
- [3] Rundgren, A., Jordan, B., Erdtman, S. **JSON Canonicalization Scheme (JCS)**. RFC 8785. (Receipt canonicalization)
- [4] Groth, J. **On the Size of Pairing-Based Non-interactive Arguments**. EUROCRYPT 2016. (Groth16 ZK proof system)
- [5] Gabizon, A., Williamson, Z., Ciobotaru, O. **PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge**. IACR 2019. (Plonk ZK system)
- [6] Maymounkov, P., Mazières, D. **Kademlia: A Peer-to-Peer Information System Based on the XOR Metric**. IPTPS 2002. (Content-addressing foundation for Filecoin)
- [7] Benet, J. **IPFS — Content Addressed, Versioned, P2P File System**. Draft 2014. (CID format used in receipts)
- [8] Adams, C., Cain, P., Pinkas, D., Zuccherato, R. **Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)**. RFC 3161. (Inspiration for cross-anchored timestamps)
- [9] NIST. **SP 800-92 Guide to Computer Security Log Management**. (Compliance baseline R+3 maps to)
- [10] AICPA. **SOC 2 Type II Trust Services Criteria**. (Compliance framework R+3 satisfies)

This document is published under CC BY 4.0. The DCS Standards Framework reference implementation is open source (Apache 2.0) at github.com/dcs-platform/standards.