

Source: b3u docs/ccs_web3.md @ 411f808 — curated 2026-08-28; edits land in the source first.

b3u.dev for Web3 — One Definition, Five Runtimes, Byte-Identical

A translation of published results for builders of consensus-critical systems

Draft 0.2 — 2026-08-28 (*0.1, same day, was the mapping alone; 0.2 adds the five-runtime demonstration of Section 6.*)

This report is addressed to web3 infrastructure builders — teams working on L2s, bridges, oracles, indexers, wallets, and node software. It maps published CCS results onto the problem class web3 teams live with, and — as of Draft 0.2 — reports a new demonstration built for this audience: a multisig custody data model, defined once and read byte-identically by five runtimes (Section 6). Section 7 states plainly what is not claimed.

1. Executive summary

b3u.dev generates, from a single grammar definition, parsers and readers for a data format in five runtimes — C++, Java, Python, JavaScript, and Rust — that provably agree byte for byte. The property has been demonstrated publicly on three formats chosen for their enterprise difficulty: HL7 v2 clinical messages, X12 EDI envelopes, and COBOL copybook records (the reports `ccs_java_hl7`, `ccs_x12`, and `ccs_cobol_data`, all at b3u.dev/docs).

Web3 is the industry that has turned this exact property — multiple independent implementations decoding the same bytes identically — into a security requirement with a name: client diversity. This report maps the published results onto that requirement and onto the adjacent workloads (bridge and oracle payloads, indexer ingestion, wallet infrastructure). Draft 0.2 adds a demonstration built directly for this audience: a **multisig custody data model** — custody records plus the propose → confirm → execute message flow — defined once, compiled once, and read **byte-identically by C++, Java, Python, JavaScript, and Rust**, with every record stored as an independently ECIES-encrypted blob whose wrong-key decode is a hard refusal (Section 6). The

report still closes with a design-partner offer, because the strongest evidence after a self-built demonstration is the same demonstration on a format a real team owns.

2. The problem class: independent decoders that must agree

Every serious web3 stack decodes the same bytes in more than one language. The node is written in Rust or Go; the frontend and tooling in TypeScript; the indexer or analytics pipeline in Python; the enterprise integrations, where they exist, in Java. Each of those components contains a hand-written decoder for the protocol's wire formats, and each pair of decoders is an opportunity to disagree.

In most industries a decoder mismatch is a bug ticket. In this one it is a consensus failure, a stuck bridge, or an exploit. The ecosystem's answer — multiple independently implemented clients, cross-checked against shared test vectors — is correct and hard-won, but it is maintained by heroics: every new format, and every change to an old one, multiplies hand-maintained decoders that must be re-verified against each other.

The alternative this report describes is to make the agreement a *generated* property rather than a *maintained* one: define the format once, generate the N decoders, and pin them to each other with a byte-identity oracle that runs in CI.

3. What CCS is, in one page

CCS (Code Configuration Space) is the technology behind b3u.dev, covered by US Patent 8,464,232 and documented publicly in the *CCS Technical Specification* and the *CCS Compiler-Compiler* overview (both at b3u.dev/docs). The pipeline, in the terms those documents establish:

1. A format or small language is described once, as a grammar.
2. The b3u.dev service compiles that grammar and returns generated parser sources plus a compact binary artifact (the SCB — syntax-controlled binary) that the runtime libraries load.
3. Consumption APIs exist for five runtimes — C++, Java, Python, JavaScript, Rust — each able to load the same binary artifact and walk the same parsed structure.
4. The correctness oracle is decompilation: each runtime's generic decompiler must reproduce the reference text form byte for byte. Agreement is not asserted; it is diffed.

Two published performance results frame what the binary form costs: the raw-binary artifact loads within approximately $2\times$ of simdjson DOM on standard JSON benchmark inputs, and a grammar-aware $O(1)$ walk outperforms simdjson's per-leaf walk on every input in that suite (*CCS Compiler-Compiler*, §benchmarks). A separate study measures the produce-once / read-many crossover for bulk loading (*MeTTa Bulk-Load Benchmark*).

4. The published evidence base

Every claim in this report traces to one of these documents, all free at b3u.dev/docs:

Document	What it establishes
ccs_java_hl7	Five-runtime byte-identical decode of HL7 v2 clinical messages; Java as the fifth runtime; JVM benchmarks; a distributed-deployment pattern
ccs_x12	The same property transposed to X12 EDI envelope syntax in one day; the obfuscation layer (XTEA and ECIES/AES-GCM) on the wire artifact
ccs_cobol_data	COBOL copybooks as compiled schemas read identically by five runtimes; a double-oracled EBCDIC/COMP-3/zoned/REDEFINES numeric-decode witness
CCS Compiler-Compiler (arXiv draft)	The self-hosting bootstrap; multi-language runtime emission; JSON benchmarks vs simdjson and nlohmann
MeTTa Bulk-Load Benchmark	Produce-once / read-many crossover for bulk structured ingestion
A Second Kind of Secret	Position paper for cryptographers: grammatical-inference hardness as a complement to cryptographic confidentiality
CCS Technical Specification	The complete reference, cross-referenced

5. The mapping onto web3 workloads

5.1 Multi-client agreement as a generated property. The trilogy's central demonstration — five independently implemented runtimes reproducing one reference decompilation byte for byte, checked by `diff` rather than by assertion — is structurally the client-diversity requirement. A team defining a new wire format (an L2 transaction envelope, a bridge message, a DA blob framing, an oracle report format) can generate its Rust, TypeScript, Python, Java, and C++ decoders from one definition and inherit the byte-identity oracle as a CI check, instead of hand-writing the decoders and hand-writing the cross-checks.

5.2 Indexers and analytics: compile once, read many. An indexer ingests the same structures millions of times. The published bulk-load result is exactly this shape: pay the structured-compile cost once, then load the compact binary form at raw-read speeds many times — with the JSON benchmark placing that load within $\sim 2\times$ of `simdjson` DOM. For pipelines that currently re-parse text at every stage boundary, the produce-once / read-many crossover is the number to check.

5.3 Keyed protection of the wire artifact. The X12 and COBOL reports document an obfuscation layer over the binary artifact: XTEA-based keyed transformation and an ECIES/AES-GCM public-key envelope, where a wrong key is a hard refusal rather than garbage output. Web3 teams are key-native — custody, rotation, and envelope encryption are daily vocabulary — so this layer needs no translation: it is at-rest and in-flight protection for structured payloads, keyed to recipients.

5.4 For the security researchers. *A Second Kind of Secret* argues a position the web3 research community is unusually equipped to evaluate: that grammatical-inference hardness describes a secret which is a *structure* rather than a *value*, complementing (not replacing) cryptographic confidentiality, and that a keyed, validity-preserving transformation on the structures of a context-free language is a primitive-design problem with no standard answer. It is an invitation, not a claim of a result — the kind of open problem `ethresear.ch` threads are made of.

6. The demonstration: a multisig custody data model, five runtimes

Built and gated 2026-08-28, after this report's first draft named the missing case study as its weakness. The domain was chosen for this audience: **custody** — the record store behind a multisig account and the message flow that moves a transaction through it.

6.1 Provenance and independence. The entity slice — safe, owner, multisig transaction, confirmation, delegate — is **our own data model**, designed after studying the schema of the open-source Safe Transaction Service at its last MIT-licensed commit (02cb9d5, 2026-02-16). Two things belong in print here. First, the license fact: that repository is dual-licensed by commit date — MIT through 2026-02-16, **FSL-1.1-MIT** (Functional Source License, converting to MIT two years per release) from 2026-02-17 — so all schema study was pinned to the MIT-era tree, and no upstream code, schema file, or data appears anywhere in the demonstration. Second, independence: this work is **not affiliated with and not endorsed by** Safe (Safe Ecosystem Foundation / Safe Labs GmbH); the name identifies the studied schema, nothing more. Every address, hash, and signature in the fixtures is synthetic (0x5AFE..., 0xA11CE...), and the gate enforces that allowlist mechanically.

6.2 The shape. One grammar defines two document forms: record (an entity snapshot) and message (a flow hop — propose, confirm, execute). The platform compiles a document set once into one compact binary artifact; five runtimes read that same artifact.

6.3 What the gate measured, in one run, every check green:

- **Five-runtime byte-identity.** C++, Java, Python, JavaScript, and Rust each reproduce the compiler's reference decompilation of the same binary artifact **byte for byte** — the client-diversity property of §5.1, held by `diff`, not by assertion. The four non-C++ runtimes do it through their *generic* SCB decompilers: zero per-format render code was written.
- **In-process equals out-of-process.** A Java service compiling the same documents in-process (through the demo's shared library, via FFM) produces the byte-identical artifact the command-line compiler produces.
- **Derived views agree everywhere.** Key-column extraction over the custody records and a flow summary over the message log (the second confirmation visibly reaching the threshold) match hand-computed fixtures in all five runtimes.
- **The storage law, per record.** Each custody record is compiled *alone* into its own blob; each blob is ECIES-wrapped independently to **two recipients** (a client key and a service key); decoding with the wrong recipient's key is a **hard refusal** — proven on every record, not asserted once. Key columns stay outside the blob, so the store remains queryable while the record contents are unreadable without the private key.

- **Red witnesses.** Malformed documents are refused; a corrupted binary artifact is never silently identical; flipping one byte inside a key column changes the derived view. A check that has never failed proves nothing — each of these was seen red.

6.4 What it took. One working session, on an unchanged platform: no compiler modifications, no new tokenizer, no new runtime code in any of the five languages beyond the ~200-line slice walkers. This is the third same-shape transposition after HL7 and X12 — the honest caveat being that it was performed by the platform's own engineers; the design-partner engagement (§9) exists to measure the same transfer on an outside team's format. We run the demonstration live on request.

7. What is not claimed

Stated as plainly as the siblings state their scope limits:

- **The demonstration is self-built, on our own model.** No L2, bridge, oracle, or client team has yet run this pipeline on a format they own; §6 shows the property on a custody-shaped slice we designed. That external case study is exactly what §9 solicits.
- **No interoperability with deployed systems.** The demo does not parse, produce, or exchange the wire formats of any running Safe deployment or other production service, and does not claim to.
- **Signatures are opaque bytes.** The flow carries synthetic signature fields verbatim; no cryptographic signature verification is performed or claimed.
- **Nothing here executes on-chain.** CCS generates off-chain parsers and readers. No claim is made about smart-contract execution, zk circuits, or consensus-client integration.
- **The published benchmarks are JSON and MeTTa, not web3 formats.** A team's own format and corpus would need its own measurement — which is what the design-partner engagement produces.

8. The good, the bad, and the ugly

The good. The property web3 already pays for — independent implementations that agree — exists here as a generated artifact with a mechanical oracle, demonstrated across five runtimes on three unforgiving enterprise formats and now (§6) on a custody-shaped slice built for this audience, with the per-record keyed-blob storage law proven alongside it; and the two capabilities

that surround it (fast binary load for indexers, keyed envelopes for payloads) are the two most common adjacent needs.

The bad. The five runtimes are C++, Java, Python, JavaScript, and Rust. Go — the language of Geth and much of Cosmos — is not among them today. Teams whose critical path is Go can still pin their Go decoder against the generated five as an external oracle, but that is a workaround, not the product. And in the spirit of the red-witness discipline §6 sells: that same discipline surfaced an end-of-input edge case in generated parsers during this build (a document truncated exactly at end of file can be accepted); it is pinned as an open item in the compiler's regression suite, fix scheduled — reported here because a test that has never failed proves nothing, and ours fail out loud.

The ugly. The demonstration is ours end to end — our model, our fixtures, our gate. The three siblings earned their conclusions the same way, but the claim this report makes to *you* is about *your* format, and that evidence does not exist yet. The cure is one design partner with a real format, and the terms below are shaped to find one.

9. The design-partner offer

b3u.dev is live and self-serve — the free tier requires no conversation. Beyond that, we are looking for one or two web3 infrastructure teams that hand-maintain the same decoder in two or more languages:

- You bring a wire format that hurts (and its test corpus).
- We generate the five-runtime decoder set and stand up the byte-identity oracle on your corpus — §6 is a preview of exactly what that engagement produces, gate and all.
- Results are published as this report's successor — or kept private; your call.
- Early-partner terms.

Contact: support@b3u.dev, or start at b3u.dev — the three sibling reports are the best preview of what the engagement produces.

© BET THREE LLC. CCS is covered by US Patent 8,464,232. The b3u.dev platform, the reading room, and the sibling reports cited here are at b3u.dev.