REVIEW SPECIFICATION / 2026

HTTP6

Complete Specification Set and NIST Technical Review Dossier

Frank Dylan Rosario; Lin Wang; Octavia Sulea; Francisco Perez; Dawn Lipscomb; Maxwell Calixto; Robert Bloodgood III

Dr. Taher Elgamal — Advisor and contributor

November 10, 2026

Review compilation 2.0 — prepared, not submitted

Architecture, mathematical sources, exact reference profile and reproducible evidence.

Prepared for technical review. Native-profile decisions and evidence are identified separately.

Contents

Part I. Review specification and release identity **3**
I.1. Purpose and requested review . . . 3
I.2. Evidence classes and profile boundaries . . . 3
I.3. What the submitted construction establishes . . . 4
I.4. Participants and responsibility boundaries . . . 5
I.5. Exact executable review profile . . . 6
I.6. Complete specification coverage and delivery map . . . 9
I.7. Reproduction and newly executed evidence . . . 10
I.8. Deployment completion decisions . . . 11
I.9. NIST alignment and a bounded demonstration . . . 12
I.10. Source precedence and preserved distinctions . . . 13
I.11. How to use the complete payload . . . 13
I.12. Reference register for this compilation . . . 13

Part II. Complete original HTTP6 architecture **15**
II. Introduction and design thesis . . . 15
II.1. The architectural question . . . 16
II.2. Names, products, and technology layers . . . 17
II.3. Trust principles and core protocol invariants . . . 17
II.4. What changes from the conventional stack . . . 19
II.5. The exact page-to-asset lifecycle: fourteen protocol phases . . . 19
II.6. The opening human proof and synchronized twin oscillation . . . 22
II.7. Session lifetime and route transitions . . . 22
II.8. HTTP6 protocol model, roles, and records . . . 23
II.9. Client/server APIs, headers, and errors . . . 26
II.10. The HTTP6 browser and WASIX deployment architecture . . . 28
II.11. The Wasmer/WASIX service stack . . . 29
II.12. Suggested source modules and repository boundaries . . . 31
II.13. Human and automated authorization families . . . 32
II.14. Implementation work packages and direct deployment path . . . 37
II.15. Operational security, monitoring, and recovery controls . . . 38
II.16. Open engineering decisions to fix in the protocol profile . . . 39
II.17. Architectural conclusion . . . 40
II. Appendix A. Vocabulary and source basis . . . 40

Part III. Executable Rosario–Wang attachment **42**
Abstract . . . 42
A. The implemented statement and private witness . . . 42
B. Geometry selection and the source profiles . . . 43
C. Private entropy as a counted circuit resource . . . 43
D. The observer’s posterior, including public code . . . 44
E. Enrollment, origin, and application authentication . . . 45
F. One-use validation and the local business effect . . . 45
G. Human origin and per-message machine proofs . . . 46
H. Efficiency and quantified experiments . . . 46
I. Reproduction and evidence status . . . 47
Source register . . . 47

Appendix A. Complete executable-envelope schema **48**

Appendix B. Payload-case acceptance register **52**

Appendix C. Schema, model, and source-code inventory **53**

Appendix D. Submission and authorship controls **54**

Part I. Review specification and release identity

I.1. Purpose and requested review

HTTP6 is proposed as a secure application-channel architecture for human, workload, and agent traffic. The Human Admission Client supplies the human-facing approval interface; Rosario–Wang supplies the private-reference membership construction; the Proof Authority, the WASIX execution sandbox, the Policy Decision Point, and the Authoritative State Plane supply instance admission, protected execution, policy, and authoritative state. The review question is whether these components can preserve bounded human-originated authority through every protected request, response, and descendant process. The complete architecture is reproduced in Part II, and the executable mathematical attachment is reproduced in Part III. [D1–D4]

This edition assembles the complete **available specification set**, rather than silently promoting a design objective into an implemented transport. It provides the full architecture, an exact account of the delivered review implementation, source-defined privacy and recovery propositions, deployment acceptance criteria, schemas, source code, and reproducible test evidence. Source manuscripts, reference implementations, and proposed native-profile decisions remain individually identified. The original files are retained separately with SHA-256 preservation records. [D1–D8]

The requested first NIST action is technical feedback and discussion of demonstration fit with the NCCoE agent-identity and authorization project. That project currently reviews feedback on a rolling basis. Its announced initial implementation use case is DevSecOps, and its summary describes evaluating existing standards with extensions, enhancements, and profiles. A bounded agent workflow over an established carrier is therefore the proposed demonstration entry point; the native HTTP6 channel remains the parallel protocol research objective. [N1–N3]

The release date is **November 10, 2026**. The current author group is Frank Dylan Rosario, Lin Wang, Octavia Sulea, Francisco Perez, Dawn Lipscomb, Maxwell Calixto, and Robert Bloodgood III. Dr. Taher Elgamal is credited separately as advisor and contributor, following the supplied revised manuscript. Historical documents keep their original bylines. Author approval, affiliation, contact details, and public-release permissions are recorded in the submission checklist rather than inferred. [D4]

I.2. Evidence classes and profile boundaries

The packet uses four evidence classes. **Mathematical result** means a proposition proved under its stated finite-model hypotheses. **Executed reference result** means an observed behavior of the identified code and test environment. **Architecture requirement** means a target behavior specified for an implementing component. **Release decision** means an algorithm choice, deployment parameter, approval, or review deliverable that must be resolved for the selected profile. These classes permit affirmative, precise claims without confusing their supporting evidence.

Profile	Definition in the supplied material	Evidence in this delivery
Review implementation	`http6-rw-review-1.0`; finite private tape, calibrated native witnesses, independently keyed HMAC envelope, local registry and SQLite	Runnable source, exact arithmetic audits, regression tests, and fresh local/loopback evidence

Profile	Definition in the supplied material	Evidence in this delivery
W1 compatibility	Proof-bearing application records carried through HTTPS; protected resource adapters and the selected inner protection	Full architecture and interface contracts; the delivered plaintext loopback harness is not labeled a completed W1 deployment
W2 managed client	Trusted workspace or native companion, controlled rendering, protected state, explicit carrier	Browser and runtime architecture plus acceptance work packages
N1 native	Independently authenticated service and client instances, transcript-confirmed key establishment, encrypted records, state and lifecycle	Complete architectural target and a decision register for the exact native suite, wire codec, and implementation evidence

The review implementation uses a new native membership word for each demonstration request and response. The broader architecture also permits lightweight record authenticators generated by a previously admitted runtime. These are two explicitly separated record-proof profiles. Neither forces a human to repeat a full login for every image or data chunk. Record class, proof profile, and inherited scope must be fixed in an eventual interoperable specification. [D1, §§6, 8.3; D2, §§2–4]

The native cryptographic schedule is computational: it derives directional keys from authenticated secret inputs and handles key updates. The exact credential-privacy profile separately consumes fresh independent private masks. A key-derivation function does not silently replace that fresh-mask premise. Public timestamps index context; they are not credited as secret entropy. This distinction governs all uses of the phrase “entropy pool” in the compiled packet. [M1, §§7–9; D1, §8.2; D3, §C]

I.3. What the submitted construction establishes

Let $X = (k, \varphi)$ be the enrolled string and private naming bijection, c_t the public membership function, and m_t the designated private mask. The source response and verification relation is

$$w_t = \varphi(c_t(k_{r(t)})), \qquad z_t = (w_t + m_t) \bmod H. \tag{R1}$$

The index $r(t)$ selects the enrolled position. The cell count H is the response modulus. Verification subtracts the actual mask and reverses the naming map, then compares the result with the enrolled membership value. Honest correctness follows by cancellation. Calibration identifies the private dictionary to an authorized decoder, and separating facets identify the symbol. The native verifier and the masks-only recovery routine are separate executable functions. [M1, §§3–5; D3, §A]

When each response coordinate has an independent uniform private mask, the likelihood of every released word is the same for every supported credential. With credential-independent public challenges, and conditional freshness over the full history,

$$\Pr[Z_e = z \mid X = x, B, G_e = g] = H^{-q_e}, \qquad I(X; \mathcal{T}_{1:N} \mid B) = 0. \tag{R2}$$

Here B is the declared prior background, G_e the complete public challenge, q_e the calibrated response count, and $\mathcal{T}_{1:N}$ the specified released history. Exactly one mask explains a fixed output for each candidate credential. Multiplying credential-independent conditional probabilities proves the sequential statement for any supported finite N. The observer's posterior remains its prior, including a nonuniform prior. The proof does not require enumerating a trillion sessions. [M1, §§4, 9]

The observation boundary includes public code and public circuit descriptors. Actual masks and enrolled references are private resources. Supplying those masks to the authorized recovery function recovers the enrolled material under calibration and separation. Acquisition of a secret-bearing runtime is therefore a different exposure experiment from acquisition of the public binary code or descriptor. Timing, errors, device signals, and active feedback are analyzed jointly when included in a stronger observation claim. [M1, §9.3; M2, §§4–6; D3, §D]

For the actual reference envelope, let s_e be the canonical statement and Z_e its native witness. A separate independent key authenticates the complete record:

$$\begin{aligned} C_e &= \{\texttt{statement} : s_e, \texttt{released} : Z_e\}, \\ \tau_e &= \mathrm{HMAC\text{-}SHA256}_{a_v}\left(\begin{array}{c} \mathrm{ASCII}(\texttt{witness-envelope}) \parallel \texttt{0x00} \\ \parallel \mathrm{canonical}(C_e) \end{array} \right). \end{aligned} \tag{R3}$$

The statement contains the actual payload, both peers, channel, direction, stream, epoch, sequence, policy and lineage identities, profile, challenge, and expiration. Changing any covered byte changes the authenticated object. Context authenticity is computational and is additional to the credential-privacy theorem. Application payload confidentiality belongs to the selected encrypted transport/record profile; the runnable local carrier deliberately displays synthetic data. [D2, §§2, 5; D3, §E; D5, `mac`, `Peer.respond`]

I.4. Participants and responsibility boundaries

The human participant approves initial scope through the selected protected admission ceremony. The admission verifier checks the enrolled relation and binds acceptance to the proposed root. A parent instance may ask the protected mint for a child only when its policy permits spawning. A child is a distinct executing principal, with its own reference and key, rather than a copy of the human's identity. The service twin checks incoming records and generates independently protected replies. The resource gate controls the actual effect. These roles are architectural responsibilities, even where the review program represents several of them inside one process. [D1, §§2–3, 8; D2, §2]

A trusted service-discovery relationship must exist before a human is asked to approve a purported service. The enrolled descriptor establishes the intended service identity; routing addresses locate it. Device evidence supplies independently appraised facts about the client key and executing instance. A processor number, interface address, or caller-supplied machine label is a context attribute rather than proof of secret-key possession. The deployed key-use interface also enforces the intended operation before generating its authenticator. [D1, §§5, 8.4; D4, §§5–7]

The confidentiality of a human's commitment does not itself authenticate a child or prevent live forwarding. Heredity is established by the authenticated grant chain, current instance possession, and independent policy gate. A forwarded authentic encrypted record may reach its legitimate endpoint without granting the forwarder permission to alter it. A separate hardware-bound profile checks the local trusted screen, protected confirmation, device key, and exact session against relayed-screen attempts. The current script exercises logical context mismatch; it does not manufacture hardware evidence. [M1, §12.4; D3, §E]

I.5. Exact executable review profile

I.5.1. Encoding and bounded inputs

The preserved program uses `VERSION="http6-rw-review-1.0"`, `MAX_PACKET=1_000_000`, and `MAX_WORD=1024`. Its canonical codec accepts ASCII-encoded JSON produced with sorted keys, separators `,` and `:`, ASCII escapes, and no floating-point values. Null, strings, booleans, bounded exact integers, arrays, and string-keyed objects are supported. Integer magnitude is less than 2^{256}; recursion depth is at most 32; the complete canonical record is at most one million bytes. Wire parsing rejects duplicate keys and any byte sequence that does not equal the canonical re-encoding. This is the program's local codec, not an assertion of RFC 8785 compliance. [D5, `canonical`, `parse_packet`]

Operation, resource, and destination are required nonempty strings of at most 256 characters in a challenge payload. Other payload data must satisfy the canonical codec. Nonces are nonnegative integers below 2^{128}. Default challenge lifetime is 120 seconds; the challenge API accepts lifetimes from 1 through 3,600 seconds. A review runtime's default private tape is 8,192 bytes per enrolled sender. The profile's complete typed envelope schema appears in Appendix A. Schema validation supplements, rather than replaces, the runtime's cross-field and state checks. [D5, `Runtime.challenge`; D6]

The complete statement is immutable once issued. The verifier retains its authoritative copy. Required field groups are shown below; the original schema fixes all field names.

Group	Statement fields	Required interpretation
Profile and event	`protocol`, `kind`, `epoch`, `event`, `nonce`	Exact profile and full event namespace; modulo geometry alone is never the uniqueness key
Peers and context	`sender`, `recipient`, `channel`, `direction`, `stream`	Match independently supplied local binding, not an untrusted header assertion
Order and time	`sequence`, `issued_ns`, `expires_ns`	Fresh current-generation position and unexpired challenge
Instance and policy	`device`, `measurement`, `policy_digest`, `lineage_digest`	Match the registered reference-model instance and unchanged admitted policy
Application	`operation_id`, `request_ref`, `payload`	Complete action bytes and, for a response, the exact prior accepted request
Membership	`profile`, `length`, `calibration_cells`, `responses`, `frames`	Correct geometry, expected calibrated count, and valid residue domains

I.5.2. Geometric profiles and entropy allocation

`ring4` uses symbols `ABCD`, two cells, two data coordinates per symbol, and two calibration entries. `rw12` uses `123456789*0#`, four cells, the twelve unequal ordered pairs in $\mathbb{Z}_4^2$, and three calibration entries. `rw40` uses the source's forty balanced triples with 36 alphanumeric live symbols and four non-enrollable pads. It uses three data coordinates per symbol and three calibration entries. The added cyclic nonce reindexing for `rw40` is explicitly labeled an engineering extension that preserves the table's separation and balance. [M1, §§6, 10; M3, §15; D2, §3]

`PrivateTape.allocate` reserves a complete event bundle under a lock, advances its cursor before releasing any values, and refuses exhaustion or a repeated event allocation. It consumes disjoint bit positions, including allocations later rejected or cancelled. The reference uses operating-system cryptographic randomness to initialize the finite tape. The exact probability audit separately enumerates the ideal distribution. Erasing buffer positions in Python is recorded as an implementation action, not an assurance of physical memory erasure. [D5, `PrivateTape`]

Profile	Enrolled length	Calibrated coordinates	Fresh mask bits per record	Trillion-record packed mask material
RW-12/4	12	27	54	6.75 TB
RW-40/4	24	75	150	18.75 TB

A request and response require two independent record allocations. The table uses decimal terabytes and excludes overhead, duplication, provisioning, and payload encryption. A 65,536-bit tape contains 1,213 complete 54-bit records or 436 complete 150-bit records. The capacity command calculates these quantities symbolically. These are resource requirements of the selected independent-mask encoding, not storage claims for every possible cryptographic profile. [M1, E4–E5; D3, §C]

I.5.3. Challenge, witness, and validation algorithms

The following sequence summarizes the actual code. The full source is included unchanged in the runnable kit.

```
ISSUE_REVIEW_CHALLENGE(binding, payload):
    validate kind, lifetime, payload and canonical encoding
    require registered sender and live ancestry
    select unused full event and next directional stream sequence
    bind exact action, peer, instance, policy, lineage and geometry
    allocate a complete fresh private mask bundle
    store the authoritative issued statement
    privately provision matching sender and verifier mask slots
    return statement and domain-separated authenticated challenge

RESPOND_TO_CHALLENGE(challenge, local_binding, intended_payload):
    authenticate the challenge under the independent instance key
    require exact local peer/channel/direction/stream and intent match
    require registered device, measurement, policy, lineage and profile
    require expected geometry, lifetime and unspent local material
    mark local event spent before computing its response
```

```
    generate calibrated native Rosario-Wang masked word
    authenticate {statement, released} under witness-envelope domain
    return the complete envelope

VERIFY_AND_APPLY(envelope, independent_binding):
    validate representation and exact authoritative statement match
    require expected endpoints, current epoch and valid deadline
    verify witness-envelope authenticator
    atomically claim ISSUED -> BURNED; refuse any repeated claim
    obtain designated private masks and check the native witness
    recheck live ancestry, committed policy and advancing stream order
    require every ancestor's complete-operation policy
    activate root, admit an approved child, validate response linkage,
        or execute/reconcile one local business operation
    record outcome; expose only the permitted public observer record
```

Authentication of the envelope precedes the claim. Native witness verification follows the claim. Thus a malformed or unauthenticated packet does not obtain a native attempt, while an authenticated complete response with an incorrect native word consumes that event and is rejected. Repeating it encounters consumed state. This ordering is a specified part of the source burn model and the actual implementation. [M1, §8; D5, `Runtime.submit`]

I.5.4. One-use state and application outcomes

The mathematical event model has RESERVED, ISSUED, and BURNED stages. The local ledger records an event as ISSUED after its private allocation and uses a committed SQLite transaction for the ISSUED-to-BURNED claim. BURNED is absorbing for the same event identity. Reopening the runtime retires unfinished issued events to `RECOVERY_HOLD`. Private references and masks are not restored from the effect database; this is controlled local restart behavior rather than a whole-machine rollback defense. [M1, §8; D5, `Ledger`]

A business operation is keyed by `(sender,operation_id)`. The first permitted request inserts its canonical payload and event reference in the local effect table. A fresh valid proof for the same operation and identical payload yields `RECONCILED`; the same identifier with different content yields `OPERATION_CONFLICT`. This establishes one local table insertion. An external payment, file write, or database service needs its own durable idempotency and recovery contract at the actual effect boundary. [D3, §F]

The local sequence rule is strictly greater than the last accepted value for that tuple of sender, recipient, channel, direction, and stream. It is not a sliding-window native transport: later acceptance can make an earlier in-flight record stale. The native architecture separately calls for bounded in-flight windows and authenticated predecessor state. The compiled specification retains that distinction rather than attributing transport-level retransmission or reordering support to the reference counter. [D1, §6; D5, `Runtime.submit`]

I.5.5. Peer grants and inherited policy

The reference `Policy` permits exact operation/resource/destination tuples and applies wildcard-capable denials. A registry grant authenticates the instance, parent reference, policy, modeled

device and measurement, and human receipt using an independent local issuer MAC. The new child must have a narrower allow set and a smaller remaining depth; each ancestor's denials and live status are rechecked. Public digital-signature verification, the richer IAM/RBAC schema, shared monetary budgets, attestation, and distributed revocation remain separately identified architecture integrations. [D5, `Policy`, `Registry`; D7]

The demo human fixture can ignite the inactive root. The root can mint `reader-v1` and read the approved report route. A successful mint creates `worker-1` with an independent machine reference and key. That leaf can read the approved resource, cannot export, and cannot mint a grandchild. Root activation records the human receipt; it does not derive child credentials from the human's secret. [D2, §2; D5, `setup_runtime`, `ignite_and_mint`]

I.5.6. Request and response contract

Every REQUEST produces a fresh native witness and envelope. Every RESPONSE uses the service's independent reference and masks. Response validation requires an accepted or reconciled prior REQUEST with matching reversed sender and recipient and the same channel. The response's payload is separately authenticated. A transport response indicating a verifier decision is not itself the modeled service-response proof. The original loopback harness sends requests through a localhost HTTP server and validates the fresh service-response proof locally. [D2, §2; D5, `run_demo`, `Runtime.submit`]

The optional carrier accepts `POST/http6-review/submit` on a temporary port bound to `127.0.0.1`. It requires a bounded Content-Length, rejects Transfer-Encoding, parses canonical bytes, and uses a handler-bound `Binding`. It returns a canonical JSON decision with HTTP 200 or HTTP 400 for handled request errors. The local client disables ambient HTTP proxy use. Only synthetic data is transmitted; no service named `reports.example` is contacted. [D5, `loopback_submit`]

I.5.7. Errors and observation records

The reference emits structured states including ACCEPTED, RECONCILED, REPLAY, EXPIRED, REJECTED, PRIVATE_STATE_UNAVAILABLE, OPERATION_CONFLICT, and RECOVERY_HOLD, with expected validation exceptions recorded by the test harness. The native architecture's proposed error families are retained separately in Part II, §9. The two sets are not asserted to be identical wire enumerations.

`observer_honest.jsonl` contains accepted public statements, masked words, and tags. It intentionally excludes enrolled words, dictionaries, raw responses, tapes, and keys, but includes the synthetic payload and metadata. Failure diagnostics and timing are separate evidence. The exact privacy theorem is applied to its declared view; unrestricted active attempts, complete Python timing, and later private-state exposure are not relabeled as part of the exact finite audit. [D2, §§2, 5]

I.6. Complete specification coverage and delivery map

The following index identifies the entire available design, code, and acceptance basis. Architecture-only rows retain their completion gates in the decision register; every row is included even when it is not exercised by the local script.

Area	Specification source	Delivered implementation or review artifact
Product boundaries and terms	Part II §§1–2	Complete source; Human Admission Client / reference architecture / HTTP6 separation
Trust invariants	Part II §3	Ten architecture invariants and local enforcement correspondence
Service discovery and first contact	Part II §5, phases 1–3	Descriptor requirements; native decision ND-01
Human and machine admission	Part II §§5–6, 13	Synthetic admission fixture and independent child references
Native authenticated handshake	Part II §§5, 8.2	Transcript/key-confirmation requirements; ND-02
Wire encoding and extension rules	Part II §8.1; Part I §5.1	Exact review codec and schema; native codec ND-03
Payload-bound records	Part I §5; Part III §E	Native RW word, HMAC binding, positive/negative payload cases
Streaming and media	Part II §§8.3, 11.2	Chunk/track requirements and performance acceptance gates
Runtime and route identity	Part II §§7, 8.4, 13.1	Logical labels/checks; hardware and route appraisal ND-05
Browser and content handling	Part II §10	Three-stage workspace/native plan; browser gate ND-09
WASIX service placement	Part II §11	Capability boundary, components, and module inventory
Hereditary policy and IAM/RBAC	Part II §13; D7	Full architectural schema plus smaller executable policy
State, retry, and recovery	Part II §§7, 13, 15	SQLite one-use and reconciliation tests; distributed gate ND-06
Offline and catastrophe recovery	Part II §13.2–13.5	Three recovery tiers; fresh epochs and custody requirements
Observer privacy and entropy	Part III §§A–D	Formalizations, exact audits, capacity reports
Security experiments	Part II §8.5; Part III §§E–H	Sixteen explicit payload cases and further acceptance plan
NIST relationships	Assurance volume Chapters 6–10	Twenty family cases, 106 CSF outcomes, classification fixtures
Review evidence and reproduction	Part I §7; Appendix B	Full runnable kit, schema catalog, manifests, logs and instructions

I.7. Reproduction and newly executed evidence

On November 10, 2026 the unchanged reconstructed validation kit was extracted and executed under Python 3.13.5 on Linux, with SQLite 3.46.1. The run passed **81 unit tests**, **16 explicit payload cases**, and **two published native vectors**. The local workflow admitted 18 native proofs and produced eight application effects. The loopback workflow performed three request/response pairs with three effects, under the carrier scope described above. The finite audit checked 2,048 binary executions, 294,912 RW-12 executions at one nonce, 2,048 two-event executions, and 864 calibrated RW-40 single-symbol recoveries. [E1]

The new run also measured 10,000 native primitive records. Its raw benchmark names platform, durations, and exclusions. It does not include the HMAC envelope, database, network, hardware, transport encryption, or private provisioning. Historical benchmark values in preserved papers remain historical and are not overwritten with a differently scoped run. The evidence directory contains complete logs, actual public records, and independently inspected SQLite effects.

From the top level of the extracted delivery:

```
python3 verify_delivery.py
python3 run_validation.py
```

The second command checks the top-level manifest, invokes the unchanged kit, and writes fresh outputs under a new `validation_runs/` directory. On Windows, use `py-3` in place of `python3`. Python 3.10 or newer and permission to bind temporary localhost sockets are required. No external network, API key, user account, or third-party package is required for this primary validation path. Restoring old proof files is not a validation shortcut: the runner creates new challenges and unused private allocations for every new trial. [D2; E1]

The expanded assurance volume also contains a separate signed-authority reference model, finite models, schemas, and an evidence-record checker. Its own dependencies and run instructions remain in that standalone directory. Its recorded original counts are preserved as inherited evidence unless a distinct new receipt explicitly identifies their execution. No Lean kernel build is asserted in this regeneration. [D8; M4]

I.8. Deployment completion decisions

The native profile reaches an interoperable release when the following assignments have concrete specifications, test vectors, and evidence. These are source-defined completion items, not silently selected algorithms or fabricated deployment results.

Decision	Required deliverable	Evidence for closure
ND-01: trusted discovery	Exact descriptor syntax, trusted issuer/bootstrap, key rotation and withdrawal	Independent first-contact, substitution, expiry, and recovery tests
ND-02: native crypto suite	Selected authenticated key establishment, identities, transcript, key confirmation, AEAD, labels and rekey	Complete composition analysis, known-answer vectors, independent endpoint agreement
ND-03: native wire format	Canonical binary codec, framing, maxima, extension rules, negotiation and failure behavior	Byte-identical encoding and adversarial parser corpus
ND-04: private-resource provisioning	Assessed private resource source, delivery, quotas, custody, retirement, and replenishment	Source assessment and leakage/lifecycle evidence for the selected exact or computational claim
ND-05: local hardware and screen	Real quote format, evidence appraisal, protected key operations, trusted display/input and pairing	Live relay, copied-runtime, wrong-device, and misleading-intent experiments
ND-06: distributed state	Consistent allocation/consumption, fencing, replica failover, rollback and durable outcomes	Fault-injection and resource-side results under the declared failures

Decision	Required deliverable	Evidence for closure
ND-07: application adapter	Exact normalization, permission semantics, data classification and idempotent effect integration	No bypass route; authorized effects and prohibited effects independently inspected
ND-08: child identity	Public issuer/signature profile where needed, templates, budgets, lifetime and revocation	Cross-verifier grants, privilege-expansion refusals, shared-budget and cessation tests
ND-09: browser	Origin model, renderer isolation, resource interception, downloads, scripts and accessibility	Browser conformance, protected interaction and usability evidence
ND-10: streaming	Chunk boundaries, ordering windows, ACK meaning, retransmission, close and congestion integration	Throughput, latency, loss, removal and rekey tests
ND-11: recovery	Surviving trust anchors, quorum where selected, epoch transition and stale-state exclusion	Whole-fleet drills, current-checkpoint evidence and measured cessation/recovery
ND-12: assessment and release	Selected NIST inventory, parameters, enforcers, actual deployment, independent reviewer, approvals	Complete requirement records, signed approvals and applicable module evidence

“Complete” in the delivery title refers to the assembled available review corpus and its explicit interfaces. A native implementation-conformance statement will name the resolved N1 profile and its evidence. This permits reviewers to evaluate the full proposal now and identify the exact remaining release contract without treating an omitted choice as an implemented feature.

I.9. NIST alignment and a bounded demonstration

The retained NIST mapping covers all twenty SP 800-53 control families and all 106 CSF 2.0 Core outcomes through source-derived mechanisms or expressly proposed integrations. It also includes separate identity, cryptographic-module, CUI, privacy, and lifecycle classifications. These are catalog/example counts. The selected system boundary and authoritative requirement inventory determine actual clause coverage, and independent assessment establishes satisfaction. Unexecuted fixtures keep empty observed-result fields. [D8]

The proposed demonstration adapts the existing reporting-worker flow into a controlled software-development task: a human admits a root, a scheduler creates distinct build and test workers, workers read approved repositories and write designated artifacts, and a protected release operation requests additional approval. The current executable still uses its reporting fixture; the DevSecOps adaptation is a proposed integration, not relabeling the test results. Proposed measurements include denied destination substitution, forbidden artifact export, restricted child minting, concurrent replay, parent fencing, and preserved outcomes after interruption. [D1, §14; D8; N2–N3]

NIST’s agent project provides an engagement route for that applied example. Formal NCCoE collaboration follows the applicable notice and agreement process. A cryptographic-module submission follows its actual module and laboratory boundary; CMVP/CAVP do not themselves test industry-protocol compliance. OLIR uses its prescribed agreement and mapping templates. The current submission is a technical contribution and review request, not a universal application or an assertion of any of these approvals. [N4–N7]

I.10. Source precedence and preserved distinctions

The exact delivered code defines executable behavior for `http6-rw-review-1.0`; its schema and tests document that same profile. The original full architecture defines W1/W2/N1 targets. The mathematical formalizations define their probability and access assumptions. The NIST mapping defines engineering relationships to selected publications. The updated IEEE paper provides the public theory, worked examples, current author credits, and cited SSL history. No source is silently used to broaden another source's theorem or implementation result.

Several distinctions particularly matter. The full architecture contemplates encrypted records and lightweight continuing authenticators; the review program uses native RW words with HMAC and visible synthetic payloads. The full policy schema supports much richer conditions than the review program's tuple rules. The original architecture presents candidate public payload-digest fields, while the later IETF candidate refines sensitive-field placement; a final native codec must resolve this choice explicitly. The later Lean registry is a source cross-reference, not a new kernel receipt. The unreadable unnumbered formalization is listed as unavailable rather than assumed identical to its numbered counterpart. [D1–D8; M4]

The source archives retain their original dates and language. Newly written submission pages identify the November 10 compilation, current routing, and new evidence. They do not retroactively change the authorship of earlier mathematical documents. Advisor credit follows the owner's supplied record; the historical SSL sources establish the historical account, not coauthor assent or approval of this submission. The release checklist records those approvals separately. [D4]

I.11. How to use the complete payload

Begin with the technical contribution in `00_submission`, then read this specification and the full architecture in Part II. Part III provides the exact mathematical and executable attachment. Appendix A contains the complete delivered envelope schema. Appendix B indexes all sixteen payload cases. Appendix C identifies the remaining schemas and machine-readable models. The full local source and instructions are in `04_validation`; the complete assurance source, appendices, examples, and independent reference code are in `05_assurance`.

The compiled review binder adds the latest IEEE theory, the payload-evidence white paper, and the 159-page expanded NIST review volume. Original source PDFs and Markdown remain independently usable. Checksums authenticate byte consistency against the supplied manifest; they do not constitute a publisher signature or a security certificate. Nothing in this packaging operation sends mail, publishes a draft, enrolls a collaborator, or commits the authors to a disclosure agreement.

I.12. Reference register for this compilation

[D1] *HTTP6: Proof-Native Web Protocol, Process-Bound Oscillation and Human-Authorized Fleet Recovery*, November 8, 2026. Full 17-section architecture; reproduced in Part II. Original Markdown and illustrated PDF retained in `01_specifications`.

[D2] *HTTP6 / Rosario–Wang — Reconstructed Validation Kit*, Kit 1.1, original core 1.0. `04_validation/HTTP6_RosarioWang_Validation_Kit/README.md`, `QUICKSTART.md`, and detailed guides.

[D3] *HTTP6: Executable Rosario–Wang Proof and Observer Demonstration*, November 9, 2026. Reproduced in Part III; original retained in `01_specifications`.

[D4] Rosario et al., *HTTP6: From Durable Credentials to Continuous, Hereditary Authority*, author and SSL-history update 1.1. Updated theory, contributor record and cited SSL history retained in `02_white_papers` and `00_submission`.

[D5] `http6_rosario_wang_demo.py`, core version `http6-rw-review-1.0`, unchanged. SHA-256: `fb5fb232e96967309ddd9439c5c902fb60876038a9fba007f84e81903f111930`.

[D6] Original envelope and fixture schemas, payload cases, and two public native vectors in the reconstructed kit; indexed in `07_schemas_and_examples`.

[D7] *Hereditary Authority from the Human Pilot Light* and `Authority_Policy_Schema.json`; full source and schema retained.

[D8] *HTTP6: Human-Originated Authority for NIST-Aligned Cybersecurity*, Expanded Public Review Edition 2.0; complete 159-page volume, modular chapters, twelve appendices, schemas, examples, models, source and prior evidence retained in `05_assurance`.

[M1] *Rosario–Wang: Consolidated Construction*, September 16, 2026. Response equations, exact verification, entropy, burn state, sequential privacy and active-experiment distinctions.

[M2] *A Unified Geometry of Private Navigation and Observable Projection*, September 13, 2026. Fibers, private slices, navigation, distributional criteria, and exposure analysis.

[M3] *Rosario–Wang: Complete Formalization of Geometric Incidence Encoding, Transcript Privacy, and Private-Reference Recovery*, numbered `(2)` attachment, September 14, 2026. Exact calibrated construction and 40-symbol family.

[M4] *Appendix B — Lean 4 Formalization Registry*. Declaration correspondence and recorded validation boundaries; no new kernel run in this compilation.

[E1] `06_evidence/regeneration_2026-11-10/validation_report.json` and accompanying logs, vectors, public records, effect databases, and benchmark. Freshly executed local reference evidence.

[N1] NIST NCCoE, *Agent Identity and Authorization project page*, rolling-feedback statement. Checked November 10, 2026. `https://www.nccoe.nist.gov/projects/software-and-si-agent-identity-and-authorization`

[N2] NIST, September 29, 2026 project bulletin; DevSecOps use case and `Ai-Identity@nist.gov`.
`https://content.govdelivery.com/accounts/USNIST/bulletins/42d04b7`

[N3] NIST NCCoE, *Summary of Comments on the Concept Paper*, standards/extensions/profiles direction.
`https://pages.nist.gov/nccoe-ai-identity/summary-of-comments.html`

[N4] NIST NCCoE, *Collaborate with Us: Technical Contributions*, formal project participation and letters of interest.
`https://www.nccoe.nist.gov/get-involved/collaborate-us-technical-contributions`

[N5] NIST, *CMVP FAQs*, protocol-versus-module boundary, SG-6.
`https://csrc.nist.gov/projects/cryptographic-module-validation-program/faqs`

[N6] NIST, *OLIR Submissions*, required agreement and mapping templates.
`https://csrc.nist.gov/projects/olir/olir-submissions`

[N7] NIST, *Crypto Reading Club*, personalized research inquiry instructions.
`https://csrc.nist.gov/projects/crypto-reading-club`

[N8] NIST NCCoE, *DevSecOps Practices project page*, currently reviewing comments; retained earlier search listing identified November 9, 2026 as its fixed-comment deadline. The rolling agent-identity route is handled separately.
`https://www.nccoe.nist.gov/projects/secure-software-development-security-and-operations-devsecops-practices`

Part II. Complete original HTTP6 architecture

The following architecture text is reproduced from the supplied November 8 specification. Heading identifiers are qualified by part for navigation. Its five original diagrams are recovered from the source PDF. Its architectural requirements and proposed identifiers retain their original scope; the executable profile is specified separately in Part I.

Citation keys within this preserved part refer to its own source register.

> **Architecture thesis.** HTTP6 pairs an enrolled Human Admission Client circuit with an authenticated Server Circuit Twin. The human establishes one live process generation; every request, response and heartbeat advances a distinct, encrypted, one-use record proof. Machine agents join by attestation and route policy. Recovery creates fresh parent and child epochs through approved machines, independent recovery TEEs or a human recovery superuser.

HTTP6 deployment	Customer/client boundary	Server placement	Channel security
W1 — compatibility	Admission web/extension workspace	Wasmer Edge HTTPS gateway, WASIX workers	Inner HTTP6 proof and encryption; outer HTTPS
W2 — managed browser	Extension view with native companion	HTTP6 gateway + protected host	Native process-state protection; selected HTTPS carrier
N1 — native HTTP6	HTTP6-aware browser or native client	Native HTTP6 listener + WASIX sandbox	HTTP6 peer handshake, authenticated encrypted records

II. Introduction and design thesis

II. Abstract

HTTP6 is the proposed next-generation web and API communication profile built on the Rosario–Wang proof construction and the human-authentication experience of the Human Admission Client. It is designed to replace the conventional separation between a long-lived transport connection and a repeatedly accepted application bearer credential with an authenticated pair of living circuit executors. A person initially proves control of an enrolled private reference through the Human Admission Client's locally rendered challenge geometry. The accepted proof authorizes exactly one client runtime generation, linked to a specified server twin, an admitted machine, an allowed route, and a declared policy. The two executors then enter **oscillation**: an ordered exchange in which each request, response, heartbeat, and selected asset block carries a fresh proof of participation in that particular channel state. The service encrypts its replies from the same authenticated channel lineage that the client uses to decrypt them; direction-separated keys, sequence values, and message-specific nonces make every record cryptographically distinct.

HTTP6 is conceived as a **full channel protocol**, not a relabeled HTTP header or an alternative name for OAuth. Its native deployment supplies peer authentication, protected setup, confidential and integrity-checked content, process-bound authorization, application-level origin policy,

bounded sandboxes, and one-use consumption throughout the transaction lifecycle. A browser user should be able to open a document, obtain verified HTML, load images and scripts, follow a link, submit a form, download a file, and conduct an ongoing conversation through the same protocol lineage. A machine agent should be able to perform corresponding operations against an enrolled server twin under machine, route, and delegation policy. HTTP6 therefore addresses the transport and authorization layers together while preserving separate responsibilities and making every security decision visible to the implementing architect.

The first deployment can be carried inside a conventional HTTPS connection to a Wasmer Edge application, so existing browsers can participate. The final **native HTTP6 channel** replaces TLS on explicitly HTTP6-aware transports, using its own specified cryptographic handshake and record protection. These profiles share the same proof and policy model; their transport packaging differs. This blueprint defines the normative target protocol, protocol state machines, API surfaces, sandbox boundary, source modules, rollout stages, and release tests. It is an architecture specification and engineering plan rather than a claim that an HTTP6-aware browser, public protocol registry, or production-safe TLS replacement has already been deployed.

II.1. The architectural question

When a customer types an address into a browser, modern clients conduct a long series of separable activities: locate the host, establish a transport, negotiate confidentiality, request a resource, receive a response, apply security policy, parse the document, discover additional resources, and repeat. A website login may occur at any point during this process. Identity and permissions subsequently flow through cookies, OAuth access tokens, session records, or signed assertions. The objects that establish permission can be replayed wherever the receiving service accepts possession or inadequate binding as sufficient.

HTTP6 changes the unit of authorization from *possession of an authentication artifact* to *live participation by a particular approved executor in a particular authenticated exchange*. The benefit is particularly clear for requests that change state: an attacker who captures the previous message has acquired a consumed observation, rather than an authority object usable for future API requests. This depends on the receiving service checking the fresh message proof and consumption record at the actual business-effect boundary.

The browser-session goal is equally important. The user should answer the original admission challenge geometry once to launch an admitted runtime. After this human admission, an active and synchronized executor pair can authorize an ongoing sequence of navigation, asset retrieval, and service messages through machine-generated per-record proofs. The person reauthenticates when a policy event terminates that particular runtime generation, such as browser shutdown, circuit termination, a changed network route, a revoked account epoch, or an explicit privileged-action requirement. The resulting model preserves a convenient continuous browsing experience **while attaching authority to each message**, rather than relying on an unexamined time interval during which a stolen cookie is sufficient.

An installed browser extension may present the first user experience inside a controlled workspace. A native companion can provide transport and process attestation outside JavaScript. A vendor can place a WASIX service behind a compatible web gateway before it deploys HTTP6-native network listeners. The protocol deliberately separates what the browser can currently delegate to extensions, what a native host must implement, and what only a browser-network-stack implementation can make first-class.

II.2. Names, products, and technology layers

The **Human Admission Client** is the customer-facing application that enrolls a person, displays the geometry challenge, obtains the person's directional response, and explains the requested origin and operation. The **HTTP6 reference architecture** is the cryptographic and operating platform (Proof Authority, Enrollment Service, Minting and Delegation Service, Policy Decision Point, Authoritative State Plane, and Consumption Ledger) that manages circuit families, custody, admission, execution sandboxes, proof checking, authoritative state, and service-to-service communication. **Rosario–Wang** identifies the proposed private-reference and proof construction used by these components. **HTTP6** names the proposed network/web protocol that transports authenticated and encrypted proof-bearing records between admitted client and server executors.

The name HTTP6 is a working protocol identifier. It is distinct from IPv6 addressing, HTTP/2, HTTP/3, and the current IETF HTTP-version namespace. Initial experiments can use an enterprise-owned scheme such as `http6://` only inside an explicitly installed and trusted resolver. The public Internet uses normal schemes until protocol registration and browser support have been designed and accepted. A proof-bearing HTTPS gateway preserves a practical deployment bridge.

A **circuit binary** is an admitted executable artifact representing a particular proof and authorization capability. A **circuit runtime** is a single executing instance of that artifact, with a measured implementation, local volatile state, a process generation, and, where supported, host-backed identity material. A **twin** is the matching remote executor responsible for one channel's state. The twin relationship is sometimes described as *entanglement*; in HTTP6 it means cryptographically synchronized state machines, not physical quantum entanglement. Each side has a matching context and independently verified role, while communication advances through directional sequence positions.

A **proof** in the opening ceremony is the person's solution to a fresh admission challenge evaluated by the authorized Proof Authority. A **record proof** is a lightweight, fresh, message-bound cryptographic authenticator generated by an admitted circuit runtime. A request and its response have separate record proofs. A **channel epoch** identifies one established twin pair; a **process generation** identifies one instance of the client process; a **route generation** identifies the normalized network path admitted by policy. Any strict route or process transition creates a new epoch and, for human policy, requires a new human proof.

II.3. Trust principles and core protocol invariants

HTTP6 has ten governing invariants. These invariants apply to the full native profile and shape the compatibility profile's application-level checks.

Invariance of server identity. The client authenticates an approved destination and its registered HTTP6 service identity before the user is invited to answer a challenge. Service availability messages are signed or otherwise protected by an enrolled trust anchor. The displayed vendor identity and origin come from that verified context, addressing deceptive cross-origin challenges and replayed advertisements.

Invariance of human admission. The geometry response is consumed for one opening ceremony bound to the exact server, channel proposal, client runtime measurement, request context, and account epoch. The process admitted after success is the same measured runtime that con-

tributed its fresh channel public material during the ceremony. A different runtime instantiates a different channel proposal and requires its own admission.

Invariance of twin identity. Client and server name both enrolled peers, roles, accepted circuit identities, and expected executable measurements. Each side confirms the complete handshake transcript. The server demonstrates its own approved executor status, rather than letting a counterfeit endpoint accept a user response and impersonate the remote twin.

Invariance of process lifetime. A human-authorized runtime is tied to a process generation. A normal shutdown closes the channel. A process crash or lost heartbeat causes the partner to retire the channel when its liveness policy expires. Relaunch creates a new generation. Human policy requires a fresh admission opening proof to create that generation; machine-agent policy can authorize a fresh attested reboot through a distinct rule.

Invariance of route generation. HTTP6's strict human profile binds the observed, verified network route to the channel epoch. Any changed source/destination endpoint, interface identity, gateway identity, or explicitly monitored IPv6 tuple suspends further application effects and starts a new admission ceremony. Network transitions therefore cannot silently carry a valid session to a substitute machine or path. IPv4, IPv6, and MAC addresses are policy attributes; cryptographic machine identity and runtime proof provide the primary origin evidence.

Invariance of per-message authority. Every accepted request and response contains a unique directional position, a canonical payload commitment, a current-channel reference, and a fresh record authenticator. The receiving twin checks the current state before releasing the content to the application. Previous receipts and copied witnesses are consumed history.

Invariance of directional confidentiality. Both twins derive compatible traffic secrets from one authenticated handshake lineage and then derive **separate keys and nonces** for client-to-server and server-to-client operations. They never encrypt two different records with the same key/nonce combination. The user describes a shared entropy pool; the implementable interpretation is a shared secret key schedule with independently mixed local randomness, domain-separated derivation, and authenticated ratcheting. Synchronization uses matching context, rather than repeated raw randomness.

Invariance of business-effect consumption. The server reserves or claims each permitted operation in durable state before executing it. The response's proof is a distinct event bound to the operation result. Ambiguous failures use same-operation reconciliation and preserve previously consumed positions, avoiding a second business effect after a retry.

Invariance of sandbox authority. WASIX/WASM executors receive only declared filesystem, network, clock, randomness, and host-call capabilities. The HTTP6 gateway keeps key custody, raw proof material, and authorization records in designated services. A compiled circuit copied from storage requires the matching admitted runtime, trusted device attributes, fresh state, and relevant human proof before participating in a protected channel.

Invariance of protocol identity. Both peers authenticate the selected HTTP6 profile, algorithms, route policy, and version. An attacker who downgrades a native connection to a weaker mode obtains a failed handshake; compatibility is enabled by an explicit vendor policy rather than by untrusted negotiation.

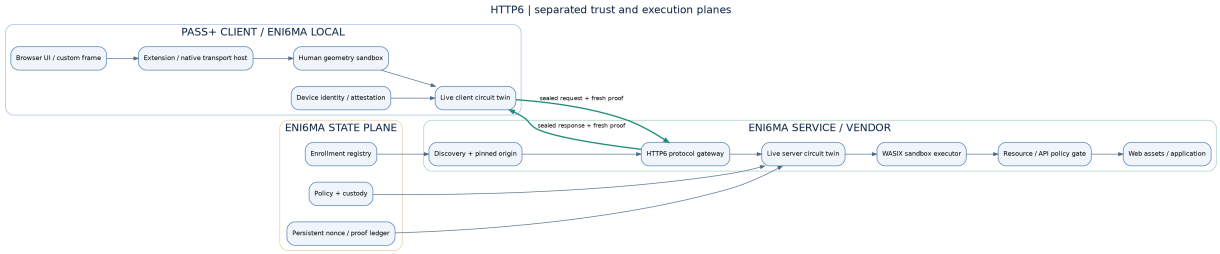

Figure 1: HTTP6 trust and execution topology, from the Human Admission Client through the Authoritative State Plane to WASIX resource services.

II.4. What changes from the conventional stack

The existing web stack uses DNS and web discovery, TCP or QUIC, TLS, HTTP request/response framing, browser origin rules, and application authorization. TLS is an essential secure-channel mechanism in today's deployments; JWT is one possible credential encoding at the application layer, while an OIDC ID token has a distinct login purpose. HTTP6's goal is to replace both *native-channel security* and *reusable bearer authority* inside a qualified HTTP6 route. This includes peer authentication, encrypted records, integrity and freshness, per-request/response proof, one-use state, and process and route lifecycle rules. A native connection should no longer depend on a JWT expiry window to authorize its next operation.

A sophisticated existing alternative, OAuth DPoP, already provides proof-of-possession binding for tokens and a unique proof per HTTP request. HTTP6 therefore should be compared against both ordinary bearer-token deployments and sender-constrained deployments. Its proposed additional design dimensions are a human-opened execution generation, admitted circuit twins, explicit process and machine identity, payload-bound bidirectional records, retained one-use state, controlled route mobility, and WASIX sandbox enforcement. A persuasive security analysis treats these as composable protocol obligations with explicit attack tests.

The resulting authorization rule can be written as a predicate:

```
ACCEPT=VerifiedPeerANDAdmittedCircuitANDLiveGenerationANDApprovedRouteANDFreshSequenceANDAuthenticRecordANDPolicyPermitsANDStateConsumed
```

A business resource receives the decrypted request only after these checks pass. An HTML renderer receives the document only after the response passes the corresponding reverse-direction checks. The same rule scales from a first page load to an API call and an agent-to-agent exchange.

II.5. The exact page-to-asset lifecycle: fourteen protocol phases

An HTTP6 implementation succeeds only if it can explain every transition between a human entering a URL and a browser displaying a verified page. The protocol is organized as a series of independently testable phases. A client can retain a long-running admitted executor while it requests multiple resources from one service twin; authorized cross-origin requests create distinct paired channels. On a new human profile, an accepted first human proof establishes the

lifetime of the client process. Subsequent assets, links, and application messages obtain individual record proofs through the active oscillation rather than asking the human to solve geometry for every image.

Phase 0: local readiness. The Human Admission Client loads the locally packaged, signature-verified client runtime. It establishes a process-generation ID, a protected memory compartment, and the device identity policy for this host. It reads trusted HTTP6 discovery roots and permitted site policies. A process launched under ordinary browser JavaScript has the assurance offered by that runtime; the higher-grade profile delegates channel state to an attested native component or suitable protected execution environment.

Phase 1: URL interpretation. The customer enters an HTTP6 resource name in the trusted HTTP6 workspace. The resolver parses its scheme, canonical authority, port, resource path, and requested operation. It rejects ambiguous encodings and checks local policy. In an ordinary Chrome installation, the address is entered in an extension-owned page or an approved native handler; an arbitrary third-party website does not thereby gain the ability to instantiate an HTTP6 peer.

Phase 2: authenticated service discovery. The client resolves a registered HTTP6 service descriptor using a pinned registry entry, verified DNS-backed service record plus independent trust, or a configured enterprise catalog. An IP address and DNS response are routing information. The descriptor binds an enrolled service identity, key or attestation trust root, supported protocol versions, policy, approved WASIX service release, discovery expiry, and supported transport profiles. The client verifies the descriptor before presenting the named service to a human.

Phase 3: availability and server proof. The server returns a fresh `SERVER_HELLO` associated with a signed or attested service identity. It contributes a fresh ephemeral public key, a nonce, an executable measurement, the accepted route rule, and its challenge of the client's proposed generation. This is the server's one-shot availability proof. The client verifies it against the discovered server identity; a copied advertisement from a different service or an unrelated process fails the expected transcript binding.

Phase 4: client circuit proposal. The client starts a new measured circuit process, creates an ephemeral secret and public value, and commits to the requested origin, intended account, client process generation, proposed route, and server hello. A TEE-backed profile attaches current device and runtime attestation. The two sides compute a canonical handshake transcript hash from both identities, offers, challenges, and agreement values. This transcript is the security context of the upcoming human proof.

Phase 5: human challenge projection. The server chooses the admitted Enrollment Service record and generates the fresh challenge for the registered circuit. The locally running circuit projects the current geometry in a trusted Admission Presenter screen. The person sees the verified server identity, scope of requested access, and current geometry; the private knowledge and mapping are used locally to form the response. The client sends a protected, transcript-bound response envelope through the participating process.

Phase 6: admission decision and one-shot consumption. The Proof Authority checks enrollment, registered circuit, selected grade, origin, current account epoch, challenger identity, and proof response. It claims the challenge before execution and commits an accepted or rejected result in durable state. Human admission belongs to the exact client process generation and

server twin committed in phases 3 and 4. A previously recorded geometry solution cannot be used to admit a different process or server context under these validation rules.

Phase 7: protected channel key schedule. The client and server mix authenticated ephemeral key agreement, fresh nonces, and the complete handshake transcript into a domain-separated key schedule. Both derive compatible traffic secrets for their roles and an independent exporter for record-proof generation. The proof establishes authorization; cryptographic key establishment supplies channel confidentiality. Either side confirms the same transcript with a keyed `FINISHED` record before application data is accepted. The server records a unique channel epoch and a policy-governed peer route generation.

Phase 8: first resource request. The browser asks for the original document (`GET /`). The active client circuit creates one `REQUEST` record, carrying the declared method, normalized route and resource ID, content preferences, operation ID, message sequence, predecessor commitment, and payload digest. It encrypts and authenticates the record, then supplies the HTTP6 proof bound to those exact bytes. The server twin verifies and consumes the message position and calls its policy enforcement point.

Phase 9: sandboxed service execution. After validation, the HTTP6 gateway invokes an admitted WASIX/WASM service with restricted capability handles. A web handler returns HTML or a manifest; an API handler performs the requested operation. Stateful effects use a durable claim and idempotency register. The application keeps its own data authorizations, and the sandbox obtains only the network and filesystem capabilities required to produce the response.

Phase 10: server proof and encrypted response. The service twin creates a distinct `RESPONSE` record. It binds the original request ID, response status, declared MIME type, content digest, sequence position, and the appropriate integrity metadata. The server encrypts the payload with a response-direction key derived from the shared channel lineage, attaches a fresh response proof, and commits its outbound state. The client verifies this distinct record before exposing bytes to its renderer.

Phase 11: page construction and asset discovery. The trusted renderer parses an admitted content type and applies the HTTP6 origin and content policy. HTML refers to CSS, fonts, media, scripts, or image objects. Each fetch is represented as its own `GET`, or as a stream of individually authenticated asset chunks. A content manifest can bind a tree of resource hashes for faster integrity checking, while every transmitted proof still advances the appropriate channel state. External origins require their own authenticated service identities and allowed channels.

Phase 12: link navigation or application event. When the person clicks a link, submits a form, or requests an API operation, the client derives an operation from the actual activated element and current document context. It canonicalizes the destination and payload, checks permission to follow that link, and advances a fresh proof on the relevant twin channel. Cross-custodian links open a separate paired channel, subject to new human approval only where the local delegation and site policy require it.

Phase 13: heartbeat, synchronization, and shutdown. Idle channels exchange bounded `HEARTBEAT` or `ACK` records, each with a unique proof, sequence, and liveness commitment. On normal exit both twins close the epoch. On missed heartbeat the server freezes new business effects until the peer completes the configured continuity policy. A changed IP, IPv6 address, network route, process, or sandbox identity in a strict human profile starts a new admission generation; recovery is a fresh handshake, never an old ticket silently carried to the new path.

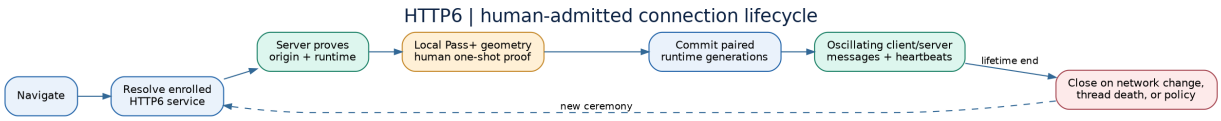

Figure 2: HTTP6 end-to-end request lifecycle, from discovery through proof and runtime-scoped communication.

II.6. The opening human proof and synchronized twin oscillation

The initial admission ceremony is a **root admission**, whereas subsequent request and response proofs are **runtime-generated continuations**. This difference makes the protocol convenient for browsing. The user establishes who may operate the local circuit and what server may pair with it; the active circuit then generates per-message cryptographic evidence as the paired state machine advances. A human need not individually answer each image or every kilobyte of a streamed video; the policy may nevertheless demand a new interactive proof for a payment, credential export, administrative change, or other high-impact action.

Oscillation is a two-party state transition. Suppose the live client is at `(epoche,C2Ssequence41)` and the server is at `(epoche,S2Csequence17)`. A client request advances only the client-to-server direction, consumes its position, and records the request commitment. The server's response advances the server-to-client direction and explicitly refers to the request. Both maintain an authenticated predecessor chain. Simultaneous messages use distinct directional chains with bounded in-flight windows; serializing the entire system across all channels is unnecessary. Record-delivery acknowledgement commits remote observation, while application-result acknowledgement confirms the business effect. These are different events.

The term *shared entropy pool* is formalized as an **authenticated common secret lineage**. Entropy originates at both participating peers during handshake and, where specified, during rekey. Each peer has the information needed to derive the same sequence of per-direction traffic keys and proof material. The peers never simply transmit a static pool of secret random bytes and repeatedly encrypt under identical nonce/key values. This approach yields synchronized cryptographic state while maintaining distinct record secrets. A server twin can operate inside a bounded WASIX sandbox while a dedicated custody/attestation service supplies its authorized key schedule and message operations.

An on-path adversary can relay bytes, so the human geometry challenge alone does not authenticate the remote origin. HTTP6 couples that human admission with independently authenticated server identity, a transcript-bound ephemeral exchange, client process identity, role-specific proofs, and final key confirmation. A transparent relay then delivers traffic to the legitimate endpoints without gaining the key material or creating a second admitted runtime; a counterfeit endpoint fails peer and transcript confirmation. A compromised authorized endpoint requires separate endpoint-hardening and resource-policy controls. These distinctions keep the anti-interception design aligned with its actual trust mechanisms.

II.7. Session lifetime and route transitions

HTTP6 treats the process generation as a live security principal. A service can mark an HTTP6 channel `ACTIVE`, `SUSPECT`, `DRAINING`, `CLOSED`, or `REVOKED`. The client and server separately maintain a last-accepted proof, generation identifier, and liveness deadline. User-visible status reflects

these states. The initial admission may open several destination-specific twins, each with its own channel epoch, while a parent policy records their common human origin.

The strict mobility policy serves applications that prefer security over seamless network switching. If the observed network tuple or required egress path changes, new protected operations pause; the runtime may inform the customer and close the previous epoch. Returning to the service from a different Wi-Fi network or a new IPv6 prefix therefore triggers a fresh human ceremony. A mobile profile can instead permit reattachment only through a separate attested rebind proof and policy; this version of the specification selects **strict re-admission for human browser sessions**. Neither source IP nor MAC alone proves the physical machine's identity; an encrypted and attested runtime supplies the primary continuity check.

On the server, a reboot or worker replacement also changes generation identity. A previous process cannot reappear merely by replaying a persistent record. A trusted server fleet can create a new twin through authenticated deployment policy. In the ordinary human-authored profile, an unexpected client restart requires a fresh admission proof. For an autonomous service, the restart decision can be delegated to attested machine policy or explicitly routed to an operator.

II.8. HTTP6 protocol model, roles, and records

The first specification edition uses the working identifiers `http6.w1` for the proof-bearing HTTPS-compatible profile and `http6.n1` for the native protocol. The names are project-defined version strings. They are not registered Internet standards. A wire implementation requires a published binary codec, test vectors, and deterministic serialization. The initial source model below is therefore an implementation contract rather than a claim of an existing interoperable endpoint.

There are seven protocol roles. The **Admission Presenter** displays the verified requesting service and the local challenge. The **Client Circuit Executor** is the measured, process-bound runtime. The **Service Discovery Authority** vouches for destination descriptors. The **Server Circuit Twin** participates in the handshake and record exchange. The **Proof Authority** evaluates initial human admission, registered agents, and retained current state. The **WASIX Application Sandbox** executes admitted web/API handlers through bounded host capabilities. The **Policy Enforcement Point** gates external data and business effects before the application receives a request. A single host may combine several roles under different credentials; the security contract names their independent decisions.

A channel begins in `NEW`, moves through `DISCOVERED`, `SERVER_ATTESTED`, `CLIENT_PROPOSED`, `HUMAN_PENDING`, `ADMITTED`, `KEY_CONFIRMED`, and `ACTIVE`. A failure moves it to `REJECTED` or `HELD`. A replay or retired epoch moves it to `REVOKED`. These states are durable enough that a process crash cannot turn a previously consumed challenge into a fresh one. A server operation has its own substate: `RESERVED`, `CLAIMED`, `VERIFIED`, `DISPATCHED`, `COMPLETED`, or `UNCERTAIN` with terminal failure categories. A server cannot infer a completed business effect solely from an authenticated transport ACK.

II.8.1 Normative envelope fields

Every protected application record carries one immutable, canonical header and one cryptographic body. The prototype schema uses deterministic CBOR or another strictly specified binary

encoding; JSON below is for readability. The sender includes an explicit record class (`REQUEST`, `RESPONSE`, `ASSET_CHUNK`, `HEARTBEAT`, `ACK`, `CLOSE`, or `REKEY`), a protocol version, channel and epoch identifiers, peer role, direction, sender process generation, normalized route ID, sequence, predecessor digest, operation ID, optional referenced request, policy epoch, content type, expiry/hold rules, and a digest of its canonical payload. The protected body consists of the authenticated ciphertext, AEAD nonce or derived nonce reference, and a fresh proof authenticator.

```
{
  "protocol": "http6.n1",
  "kind": "REQUEST",
  "channel_id": "opaque-channel-id",
  "epoch": 8,
  "direction": "client_to_server",
  "sender_generation": "opaque-runtime-generation",
  "sequence": 42,
  "previous_commitment": "hash-of-prior-directional-record",
  "operation_id": "unique-operation-id",
  "route_id": "admitted-route-identifier",
  "method": "GET",
  "resource": "/documents/report",
  "payload_digest": "hash-of-canonical-plaintext",
  "policy_epoch": 3,
  "ciphertext": "AEAD-protected bytes",
  "record_proof": "fresh-directional-authenticator"
}
```

A parser rejects duplicate fields, unknown critical extensions, out-of-range lengths, reinterpreted Unicode, ambiguous path separators, and noncanonical encodings. Protected content remains opaque to intermediaries, subject to deliberate public routing metadata. Compression, streaming, and partial reads have versioned definitions; compressing secret-dependent content alongside attacker-controlled content can create an information side channel and therefore requires its own reviewed profile.

II.8.2 Cryptographic key schedule

Initial authenticated key establishment derives a transcript secret from the approved ephemeral agreement. The service identity is independently authenticated and the client's attested ephemeral key is bound to the same transcript. A possible conventional composition uses a well-reviewed KEM/ECDH construction, HKDF, an approved AEAD cipher, and a separate MAC construction. HPKE (RFC 9180) and TLS 1.3 (RFC 9846) provide relevant engineering models for key establishment and authenticated channels; HTTP6's native use requires a separate exact composition rather than assuming these standards supply an HTTP6 wire format.

The shared lineage can be modeled as `Kroot=KDF(ephemeral_secret,transcript_hash,fresh_context,enrolled_identity_binding)`. Separate labels derive `Kc2s_encrypt`, `Ks2c_encrypt`, `Kc2s_proof`, `Ks2c_proof`, and distinct rekey materials. Each record uses a unique `(epoch,direction,sequence)` derivation; the resulting AEAD nonce remains unique for every invocation of the selected cipher. The per-record proof authenticates the canonical header, ciphertext digest, operation commitment,

previous accepted record, and live runtime generation. A proof accepted at position 42 is different from position 43, even for repeated plaintext, and a server proof cannot substitute for a client proof because its direction-specific keys differ.

The **entropy pool** is defined as coordinated, cryptographically derived state, rather than two processors reading identical exported random bytes. The clients' fresh contributions are mixed into the handshake and the derived secret schedule. Regular rekey uses the authenticated prior ratchet and a fresh contribution according to a specified hybrid derivation. The KDF can be deterministic because its secret inputs have sufficient independent entropy; proof codes, a human-entered secret string, a timestamp, and a public challenge are excluded as the sole inputs to encryption strength. Historical keys are erased from active memory according to the forward-secrecy and retention policy.

A design asserting unconditional or information-theoretic transcript privacy must separately prove that the exact observable record—including headers, sizes, timing, failures, and rekeys—satisfies its stated distribution assumptions. The implementation here uses concrete authenticated cryptography for channel protection and retains the private-reference construction for human admission. This makes protocol roles precise without conflating a transcript theorem with all properties of a deployed web transport.

II.8.3 Request and response proofs

For a client request, the policy engine authorizes the tuple `(owner,service,runtime,route,method,resource,canonical-payload-digest,sequence,epoch)`. The server verifies both ciphertext integrity and the proof-circuit record authenticator under the named current state before invoking a handler. For a server response, the client verifies `(channel,referenced_request,server_generation,response_code,response-digest,sequence,epoch)`. This creates symmetric authenticated participation: every direction proves that its sending twin controls the state of that direction.

For large media, a stream has a named parent operation and a bounded stream policy. Every chunk carries its own authenticator and sequence; video frames may map to multiple protected chunks, and a conference may have one committed track manifest plus per-packet protection. The cryptographic record proof can be a short MAC tag because the strong human proof authorized the parent executor, while the continuously advancing key schedule now proves active machine participation. The design does not require replaying the full human geometry per frame. Bandwidth and CPU budgets determine the selected record size and whether selected packets are encrypted individually or grouped into authenticated segments.

An API with write effects adds a persistent operation-consumption register. If a response is lost after a payment succeeds, the client queries the status of the **same** operation through a fresh authenticated record. The server returns the previously committed result and prevents a duplicate charge. This is the distinction between at-most-once proof acceptance and exactly-once business effect: the latter is achieved through idempotency and application reconciliation, not solely through a cryptographic MAC.

II.8.4 State, identity and routing

An HTTP6 route descriptor defines the authenticated destination service, protocol version, port and network transport, canonical origin, allowed path templates, approved destination iden-

tities, permitted IP/subnet policy, and the runtime's machine identity constraints. Remote machine identification uses attestation or hardware-backed keys. A MAC address ordinarily resides at a local link layer and is absent from routable Internet packets; it is a useful enrollment policy attribute when a trusted local agent reports it, rather than a remote cryptographic identity. An IPv6 address may identify the observed connection and can be used as a strict mobility condition, while cryptographic service identity remains the protection against masquerading.

A strict route change triggers `HOLD_ROUTE`, closes the previous channel after verifying the live peer state, and issues a new handshake and admission as required. For user-facing sessions, the selected policy mandates human reentry on process or network transition. For machines, an attested restart policy can authorize a new channel epoch after the service confirms the registered hardware, application measurement, and approved route. Neither policy reuses a previously recorded proof position.

II.8.5 Threat-by-threat security contract

Packet interception: The intercepted record contains a spent sequence position and encrypted content. A different sender lacks the matching session secrets and runtime identity; the resource checks current consumption before processing.

Executable/circuit theft: A copied WASM binary is only an artifact. Protected opening uses the machine and process admission record, current circuit epoch, custody policy and fresh proof context. Copying the program to another host produces a different runtime generation whose attempt to participate is rejected by the paired service.

Live man-in-the-middle: A relay may forward ciphertext, but channel identity, transcript confirmation, and authenticated ephemeral establishment prevent it from silently creating an independent authenticated twin. In-band phishing and a compromised legitimate endpoint remain addressed by verified destination presentation, endpoint isolation, and operation-specific approval policy.

API key theft: A legacy API key may initially remain behind a proof-gated capsule. The eventual native HTTP6 service authorizes operations through the current record proof and local policy, removing the need for a reusable bearer key at that selected resource.

Service impersonation: The server's identity and attested binary are verified before the person responds. An unauthenticated availability answer cannot select the policy or redirect the proof to a counterfeit recipient.

Reboots, lost networks, and stale backup: A retired generation and consumed nonces stay recorded in independent retained state. New process and route generations enter an admission flow, even when the same computer returns to the network. Operators can reinstate an approved fleet through the disaster recovery root procedure defined below.

II.9. Client/server APIs, headers, and errors

In the compatible transport profile, the outer HTTPS endpoint accepts a protocol-versioned opaque envelope, such as `POST/http6/v1/records` or a bidirectional WebSocket where supported. These are **proposed interface examples**, with exact names fixed by the implementing service and published SDK. The outer request performs no application action by itself; its protected

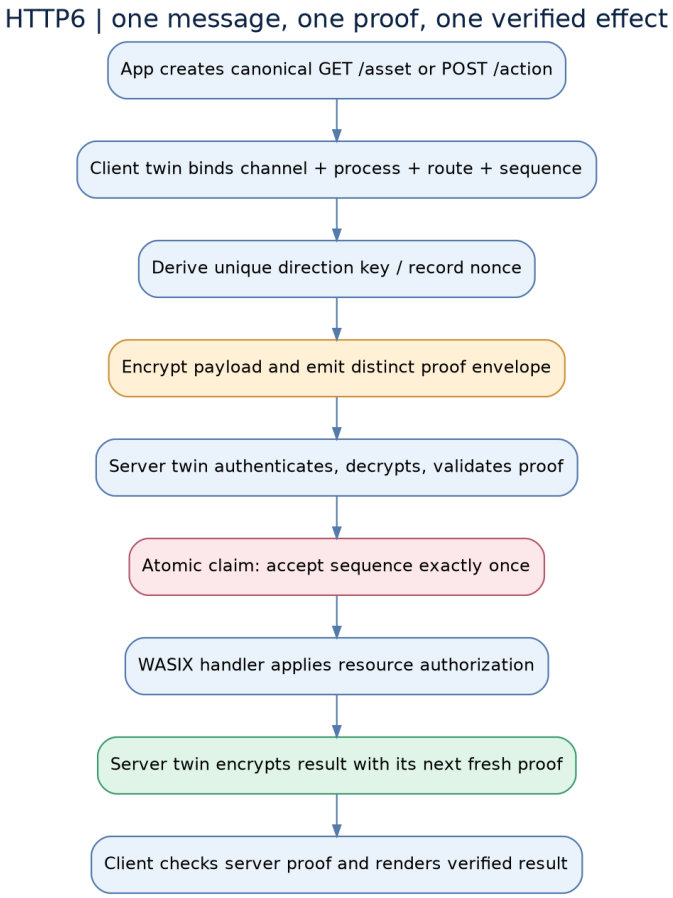

Figure 3: The HTTP6 record exchange: separate directional proofs and synchronized encryption state.

HTTP6 payload is parsed and authorized by the twin service. Ordinary HTTP status codes can communicate transport failure while the inner HTTP6 response carries the business result and proof.

In the native profile, `http6://service.example/path` selects an HTTP6-native client/transport. A future custom browser could handle the scheme and its native secure channel directly. An installed companion can register a local scheme handler and render through a dedicated browser workspace, subject to the platform's scheme registration and security model. The encrypted record framing is transport-agnostic at the application layer: reliable byte stream, QUIC-like datagrams, or a constrained message bus may carry its envelopes after the security handshake is established.

Recommended error families are `PEER_UNTRUSTED`, `ATTESTATION_FAILED`, `HUMAN_PROOF_REQUIRED`, `RUNTIME_GENERATION_CHANGED`, `ROUTE_CHANGED`, `EPOCH_RETIRED`, `SEQUENCE_CONSUMED`, `AUTH_TAG_INVALID`, `POLICY_DENIED`, `STATE_UNAVAILABLE`, `REKEY_REQUIRED`, and `OPERATION_UNCERTAIN`. Errors are authenticated and minimized so a remote unauthenticated caller cannot probe private enrollment data. Structured audit events record the allowed state transition while raw proof responses and custody keys remain within their designated execution service.

II.10. The HTTP6 browser and WASIX deployment architecture

HTTP6 needs a path from a human's normal browsing habits to a controlled execution environment. Its client strategy has three stages, each with a sharply defined capability boundary. The same protected message format and proof rules operate across the stages, so an early vendor can build an HTTP6 service without waiting for the entire public browser ecosystem to implement a new transport. The user-facing Human Admission Client retains the visible identity and proof experience; the server side of the HTTP6 reference architecture hosts the cryptographic channel, the Policy Decision Point, the sandbox, the Minting and Delegation Service, and the Consumption Ledger.

II.10.1 Stage one: HTTPS-compatible HTTP6 gateway

The first service deployment is the most attainable: an existing website or API operator deploys a new Wasmer-hosted HTTP6 gateway that uses a standard HTTPS listener. The browser opens the admission workspace using HTTPS, locally packages its circuit runtime, completes the accepted admission ceremony, then sends and receives encrypted HTTP6 frames to the gateway. Every inner request and response contains a fresh record proof. The outer TLS endpoint remains the compatibility transport for existing browsers and Wasmer Edge. The site can migrate high-value APIs to the proof-enforcing gateway before adopting a native HTTP6 transport.

The gateway handles discovery, protocol negotiation, rate limits, request-envelope validation, and routing to protected execution. The paired server twin runs under a selected WASIX/WASM runtime with bounded capabilities, and the proof authority/ledger sits in separately authenticated services. The vendor may keep existing content and database backends behind that gateway. Its only application-facing change is to expose a well-defined resource adapter that consumes a verified HTTP6 operation context.

The gateway must verify the inner proof before accessing a vendor backend. A request body copied from an earlier HTTPS exchange, even if byte-identical, is bound to a spent HTTP6 sequence and cannot trigger another effect. Existing login sessions can coexist during migration; the protected route enforces proof-first authorization regardless of a cookie's presence.

II.10.2 Stage two: extension-managed HTTP6 workspace

A Chrome extension can create an HTTP6 **workspace page** with the Human Admission Client visual identity and managed navigation controls. The user enters an HTTP6 resource address into the workspace. The extension mediates all navigation and resource fetching for that workspace through its controlled channel implementation. A remote HTML document can be displayed using a restricted renderer, a sandboxed view, or a trusted component model. URLs within the controlled view are parsed and rewritten into new HTTP6 records rather than becoming unrestricted network accesses from remote page scripts.

The extension's declared network-request API can block, redirect, or modify certain request and response headers. Its rules alone do not amount to replacement of the browser's TLS stack, nor do they provide complete control of arbitrary browser-fetched content. Chrome's `declarativeNetRequest` applies at specific browser network lifecycle stages; sensitive request bodies, response

bodies, and cross-origin page execution need other controlled components. A production extension therefore uses an explicitly trusted HTTP6 view, a packaged runtime, and optionally a native companion for its proof state and transport. A content script running in a merchant page is an untrusted input surface, with a validated bridge to the extension service worker and protected host.

A packaged extension must obey applicable installation and code-loading policies. Manifest V3 disallows running remotely hosted executable JavaScript and WASM in the extension's own execution contexts. Remote website content is treated as data or rendered in a separately specified, properly isolated origin; downloaded executable circuit material follows its permitted distribution profile. The architecture can support managed enterprise builds where selected capabilities require separate permissions, release governance, or a native component. Every renderer and compiled artifact has a signed release identity.

The extension's image, font, stylesheet, media, and document requests are all issued by the HTTP6 client and checked under the active twin. A user click is captured as an application action, not reinterpreted from arbitrary cross-origin script. An `http6://` link remains within the HTTP6 workspace unless the user explicitly selects an external web destination, in which case the appropriate normal browser security and origin rules apply.

II.10.3 Stage three: native browser transport or companion

A first-class native HTTP6 client opens its own network transport to an HTTP6-aware service, performs the enrolled handshake, and renders the verified response. It may be a custom browser, a managed browser fork, a native operating-system helper with an approved browser integration, or a dedicated desktop application. A Chrome extension can communicate with a registered native host through native messaging, subject to explicit manifest permission and allowed extension identity. Such a host can handle protected process state and transport without asking ordinary JavaScript to become a general-purpose network stack.

The native channel replaces TLS **for that connection** by implementing the same engineering responsibilities: authenticated peers, encryption, integrity, replay protection, secure key update, downgrade resistance, and error handling. Each request and response additionally carries the HTTP6 per-message proof and business authorization. A reverse proxy can simultaneously serve normal HTTPS and native HTTP6 to different client populations. Public web links, third-party APIs, package downloads, and provider integrations retain their defined transports until they adopt the new endpoint.

An operating-system network interceptor that silently rewrites arbitrary browser traffic would introduce origin confusion, protocol compatibility problems, and a large trusted code base. The preferred initial implementation is a transparent user-selected workspace with explicit enrollment, origin displays, and clear navigation semantics. A fully native browser can later give HTTP6 the same first-class URL handling that ordinary HTTP and HTTPS enjoy.

II.11. The Wasmer/WASIX service stack

The HTTP6 server is a small **protocol-facing service**, paired with a stateful **proof and authorization plane**. An organization deploys its HTTP6-capable endpoint, registers its identity and policy, and supplies handlers that accept approved operations. The original web stack can remain behind this layer. On Wasmer Edge, an ordinary app package supplies an HTTP server

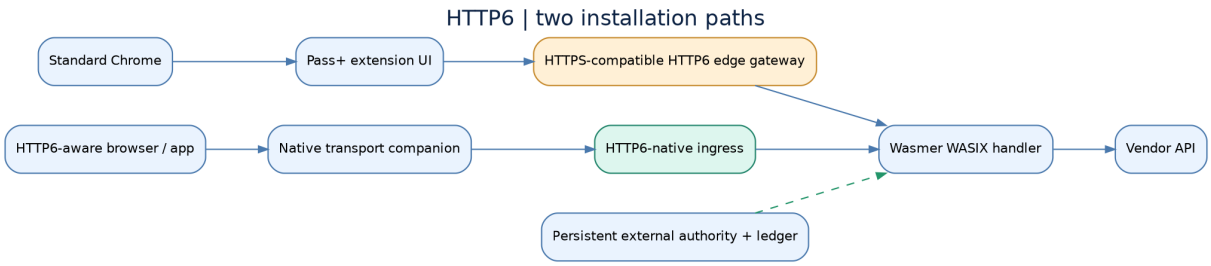

Figure 4: Deploying HTTP6 through an HTTPS gateway, a trusted extension workspace, or a native transport.

through Wasmer's deployment model. Its instances are ephemeral and may be replaced, so channel state, consumed proof positions, account epochs, business idempotency, and recovery journals must be externalized into appropriately durable storage. A process restart must be visible as a new server-generation or controlled backend reconnection, rather than quietly reusing a lost in-memory proof chain.

The service package has six primary components. The **Transport Gateway** accepts compatible HTTPS envelopes and, in a separately qualified installation, native HTTP6 records. The **Discovery Registrar** signs or references admitted service identity, route profiles, and currently approved release identities. The **Server Circuit Twin Runtime** runs only admitted binaries and emits/consumes channel proofs. The **Policy Engine** checks account, machine, route, operation, and customer entitlements. The **Custody and Attestation Adapter** accesses protected recipient material, validates machine evidence, and supplies allowed cryptographic operations. The **State and Reconciliation Store** commits sequence positions, human admission, fleet epochs, workflow results, and disaster-recovery lineage.

WASIX provides the extended WASI runtime environment and host integration appropriate to an application needing operations beyond pure computation. The deployment must select which filesystem, clock, randomness, network, process/thread, and host-call capabilities each guest can use. A sandbox is an enforcement boundary only when the host restricts its imports and protects secrets outside the guest's readable memory. Separate workers and resource budgets constrain malformed inputs and executable behavior; syscall-level filters and operating-system confinement supplement the WebAssembly boundary where deployment requires them.

Wasmer's HTTP app runtime and a native socket-listening service have different capabilities. Current Wasmer Edge documentation describes HTTP-server based app deployment and stateless, ephemeral instances, with generic TCP/UDP server support described as a future expansion. The first hosted HTTP6 product therefore uses HTTP over the platform's ingress, while native network transport can run in a separately deployed WASIX/native service or an independently managed server. This avoids making an unsupported platform feature a prerequisite for the pilot.

II.11.1 One URL, one request, one sandboxed execution

A vendor's `GET/reports/latest` endpoint registers an HTTP6 resource handler. The client performs the opening ceremony, then sends a sealed request with its message proof. The gateway

validates framing, uses the channel's admitted twin to verify the record, and asks the policy engine if this user/runtime/route can read that resource. It invokes the appropriate WASIX handler with a temporary read-only data capability and a bounded output budget. The handler returns an HTML document and a signed or committed asset manifest. The server twin protects the response, and the client releases only verified bytes to the renderer. Every subsequent asset fetch follows the same path.

For state-changing requests the gateway calls the business effect controller only after the operation is atomically claimed. This permits a conventional database or payment service to remain in place while the HTTP6 gateway replaces the reusable credential as the initiating authority. Retrying a lost response returns the outcome of the same operation rather than replaying the effect. Resource ownership and transaction limits are still checked by the consuming application.

II.11.2 Media and large streaming flows

Interactive media uses sustained twin channels. At channel creation, the participant authorizes a conference, stream role, track policy, and maximum resource budget. A centralized conferencing server can distribute a track to multiple enrolled destinations through independent HTTP6 edges, with each recipient's encrypted frame proof tied to its own channel. A multiparty mesh creates individual pairwise channels or a group state protocol with membership epochs and sender-specific keys. HTTP6's first specification selects pairwise delivery for simplicity and defines a separate group-key management profile for high-scale mesh conferencing.

Each protected media record carries a unique authenticator, sequence number, and stream/track identity. Key ratchets can advance every chunk or at fixed boundaries with a record-unique derivation. Application latency, jitter buffering, retransmission and packet loss handling must be measured: a sender that loses a packet must not create an uncontrolled proof replay; the receiver can detect missing positions and request authenticated recovery. The conference server is itself an admitted service with rights to forward only its specified objects. A shared public circuit attestation can establish membership eligibility, while each active channel keeps its distinct private proof keys.

II.12. Suggested source modules and repository boundaries

The architecture's source layout is proposed as a set of independently buildable, versioned packages. `http6-protocol` defines canonical binary envelopes, version negotiation, transcript hashing, error classes and test vectors. `http6-crypto` implements the selected standard KEM, HKDF, AEAD and record MAC schedule behind a reviewed interface. `http6-discovery` validates registered service descriptors and trust roots. `http6-client-core` implements the channel state machine, process generation and ProofCheck integration. `http6-admission-ui` hosts human geometry display, origin verification and session status. `http6-extension-workspace` handles managed view, link navigation, resource fetching and permissions. `http6-native-host` supplies protected process state and transport adapters. `http6-wasix-gateway` implements ingress and proof-enforcing handlers. `http6-runtime-twin` hosts paired circuit execution with bounded imports. `http6-policy` evaluates human, machine, route and operation rules. `http6-ledger` stores consumed positions and business idempotency. `http6-fleet-recovery` manages parent/child epochs, recovery TEE identities, human-superuser ceremonies and bulk restart orchestration. `http6-test-harness` contains loopback and fault-injection acceptance suites.

A release manifest binds each package to an exact source commit, compiled artifact digest, policy schema, capability import list, and test receipt. The browser and server runtimes can implement the same logical protocol in different languages while sharing byte-level message fixtures. Source changes to the proof engine, guest-host interface, cryptographic key schedule, or migration schema require an explicit version and interoperability test. Credentials for publication, authority, and runtime secrets belong to their designated host services rather than public source packages.

II.13. Human and automated authorization families

HTTP6 separates a **human admission circuit** from an **autonomous service circuit** while preserving one trust model. The human circuit opens a verified admission challenge, admits a particular local process, and establishes a scoped parent authorization. The machine circuit enters the corresponding oscillating relationship using enrolled hardware evidence, measured execution and a delegated policy. A fleet can include thousands of child circuits, but each child possesses a distinct identity, route, operation scope, and live proof state. One successful human superuser ceremony can approve a policy tree for those children; it does not produce a reusable master bearer credential that any child can copy and wield elsewhere.

A parent policy declares the maximum number of descendants, allowed service classes, approved machine inventory, process measurements, origins and ports, network route policies, permissions, transaction budgets, lifetimes, and human reapproval triggers. Child generation is a controlled authority operation. The newly authorized scheduler requests a child identity from the Minting and Delegation Service, the target machine presents its attestation, the Minting and Delegation Service checks the child's requested policy against the parent, and the scheduler issues a one-shot bootstrap invitation addressed to that exact child runtime. The child constructs a fresh HTTP6 twin with the remote service. A child cannot mint a more privileged sibling from its own local policy unless the parent explicitly delegates that operation.

This produces a hierarchy of **root human admission** $\rightarrow$ **fleet authorization** $\rightarrow$ **machine-specific child issuance** $\rightarrow$ **process-generation admission** $\rightarrow$ **channel-specific oscillation** $\rightarrow$ **per-message proof**. The human is the authority for the selected task set. Individual automated agents become authorized participants rather than containers for a common long-lived API secret. Every child can be suspended or retired by advancing its policy or fleet epoch. A fleet-wide emergency shutdown retires the parent and all descendant admission generations according to the enrolled revocation policy.

II.13.1 Machine attributes and the qualified path

An enrolled automated machine has a verified device key, attestation reference values, an approved runtime measurement, a current boot counter or equivalent monotonic identity, and an expected workload. The policy can also record network IP ranges, named IPv6 destinations, interface MAC identifiers, gateway identities, selected VLAN, data center zone, and route templates. These network features enforce placement and routing rules; the machine's attestation and private keys prove its stronger identity. The remote server checks current network observations and service-origin authentication before accepting each new channel epoch.

A machine reboot creates a fresh process generation and fresh channel keys. Under `attested_auto` policy the machine can rejoin automatically if the registered hardware and runtime pass attestation, its route satisfies policy, its fleet epoch is current, and the authority grants a new bootstrap. Under `human_on_reboot`, the machine sends a request to an administrator enrolled in the Human Admission Client and remains held until an authenticated approval creates the new generation. Under `break_glass_only`, even a properly attested machine waits for the designated emergency operator ceremony. These policies let low-risk infrastructure resume quickly while giving high-impact financial, custody, and administrative services direct human control.

II.13.2 Recovery authority tiers

The architecture uses three distinct recovery tiers, each creating fresh authority rather than reviving old sequence positions.

Tier A — ordinary attested reboot. A machine loses power, a cloud instance is replaced, or a WASIX guest exits. Its enrolled device and orchestrator supply fresh attestation; the Minting and Delegation Service verifies code measurement, route and fleet epoch; the service creates a new channel and starts its oscillation. Current event and consumption state come from durable stores. The procedure restores approved automation with minimal operator intervention.

Tier B — independent recovery TEE. An organization installs a physically and operationally separate recovery machine with hardware-backed keys and pre-enrolled, narrowly scoped disaster authority. The machine might be a solar-powered laptop with independent battery storage and an alternate network path. Its **power autonomy** and **communications autonomy** reduce common-cause failures affecting the primary data center. The TEE/HSM or equivalent protected runtime stores a designated recovery capability and monotonic state; solar power by itself supplies availability rather than cryptographic trust.

On a primary-site outage, the recovery machine attests its own identity and current code, reads the latest independently retained checkpoint or signed recovery policy, and requests a new disaster epoch. It enumerates the permitted machines and services, applies the authorized restart policy, and mints **fresh, machine-bound child bootstrap ceremonies** for the replacement fleet. If the alternate network is available, it coordinates new HTTP6 twins directly with reachable services. If all outside network paths are down, it can prepare sealed, signed recovery plans and local authorizations for later delivery to pre-enrolled recipients, with a defined validity window and monotonic epoch. Actual remote services become active only after they can authenticate the recovery authority and reconcile the currently accepted state. Offline work therefore preserves a controlled recovery plan rather than pretending to complete an unreachable remote authorization.

Tier C — human-superuser restoration. If the independent recovery TEE is unavailable, a designated administrator performs an enhanced admission ceremony using a pre-enrolled disaster-recovery route. The recovery policy may require two authorized humans, a threshold of protected shares, a sealed offline recovery kit, or a physical console at the surviving trust root for high-impact installations. The operator authenticates through locally verified geometry against a known recovery authority and approves a proposed disaster epoch, destination set, and bounded child-family policy. An accountable recovery signer commits the transition and distributes fresh authorization seeds or recipient-addressed bootstrap records to the surviving machines. This is **one human proof that authorizes a school of controlled progeny**, followed by individual machine and route admission for each child.

When a central authority and every enrolled recovery appliance have been destroyed, the installation follows its **root-of-trust reconstitution procedure**. The organization must possess a surviving pre-registered recovery trust anchor, offline protected credential material, or an explicitly governed replacement-root ceremony. If all trust-anchor material has been lost, a new root identity must be established and distributed through authenticated out-of-band administration before other machines can trust it. Human knowledge alone cannot make a destroyed signing key reappear. A sound disaster plan defines this material before deployment.

II.13.3 Disaster recovery invariants

An emergency restart **increments a recovery epoch**, independently of ordinary channel counters. Every old runtime generation is retired or suspended, and no old nonce becomes available again. The recovery ledger records the operator and recovery machine attestations, approval policy, source checkpoint, affected service families, new epoch, selected recipients, generated child bootstraps, acknowledgements, and first valid request proofs. Every child receives a unique new process and channel generation. The system can therefore restart a school, a cluster, or a data center from one human-authorized event without circulating a reusable fleet password.

A disaster policy defines exactly when a child may run and which duties remain held until additional review. For example, a fleet of web-cache agents may restart automatically; payroll settlement and credential exports may require a second human; signing authority may require quorum. A compromised service can be omitted from the replacement family even if it previously belonged to the fleet. Parent authorization never overrides the application owner's transaction limits or the service's own policy enforcement point.

Failure scenario 1: local utility power loss. The main data center loses electricity. Primary twins time out. The independent recovery machine remains available on its battery/solar supply, and its trusted ledger reference identifies the latest accepted state. When replacement workers boot, they enter a fresh attestation and per-machine admission sequence; stale proofs fail the epoch check.

Failure scenario 2: primary fiber outage with alternate communications. The main connection fails, but the recovery TEE has a cellular, satellite or separately provisioned private connection. The recovery authority verifies the alternative route against its own explicit policy, creates a disaster channel epoch, and issues targeted child bootstraps to reachable sites. Human browser clients with a strict route policy still require new human admission after route change.

Failure scenario 3: independent TEE process crash. A watchdog detects that the backup authority process terminated. Rebooting its operating system alone gives it no continuing channel proof. The protected machine identity and current recovery policy permit a fresh attested activation only under its designated tier-B rule. Failure of the machine attestation, tamper state or recovery material routes activation to the human-superuser tier.

Failure scenario 4: total loss of backup recovery devices. A designated administrator opens a separately registered human recovery entry point, proves control through the Human Admission Client, participates in any required co-approval, and establishes a new parent fleet epoch through the surviving root trust. Each child subsequently creates its own keys and proof stream, allowing hundreds of services to return under one approved organizational recovery decision.

Failure scenario 5: stolen child WASM circuit or cloned virtual machine. A copied binary or VM image reaches a different server. Its attempt to use the old circuit identity is bound to the previous machine, process generation and epoch. The service requires fresh machine evidence and route membership. Copying historical witness outputs provides consumed sequence positions. The cloned process must be separately admitted before it can participate in an authorized twin pair.

Failure scenario 6: stale database restored from backup. A previous ledger snapshot appears to forget spent proof positions. The service compares its state with independently retained witness/checkpoint or disaster-epoch evidence and holds operations when current state cannot be established. It resumes after state reconciliation or fresh authorized epoch transition. This is essential because runtime-bound proofs still require retained replay history to make consumption meaningful.

Failure scenario 7: malicious operator requests a huge fleet. The root policy limits delegated child count, allowed service images, destinations, resource budgets and permitted operations. A request exceeding its policy enters held review. The event is logged under the human identity and epoch, while each granted child remains independently auditable and revocable.

II.13.4 Recovery UI and operator workflow

The admission recovery console should present a compact disaster summary: affected fleet, unavailable service count, current recovery authority, machine inventory, last verified checkpoint, pending route changes, required approvers, and projected child instances. The operator selects a recovery policy, sees each privilege class, and approves the creation of a new fleet generation through the admission challenge. The console displays the new parent ID, issued children, successful attestation, channels established, held exceptions, and any incomplete business effects requiring reconciliation.

The experience distinguishes **authorization**, **deployment**, **channel establishment**, and **business readiness**. A human may approve the fleet and still have downstream workers unavailable because they lack power, accepted software, route access or a valid checkpoint. Each level is recorded separately. An offline approval package is sealed to its intended recipients and opened only by the authenticated machine, with the current policy epoch enforced when connectivity returns. A delayed package from an older epoch is refused.

II.13.5 Suggested policy profile

```
schema: http6.fleet-recovery.policy.v0
fleet: research-datacenter-west
root_admission:
  mode: human_admission
  roles: [fleet_superuser]
  approve_child_policy: true
  max_children: 512
  high_impact_actions: second_human_approval
ordinary_machine_restart:
  mode: attested_auto
  require_measured_runtime: true
```

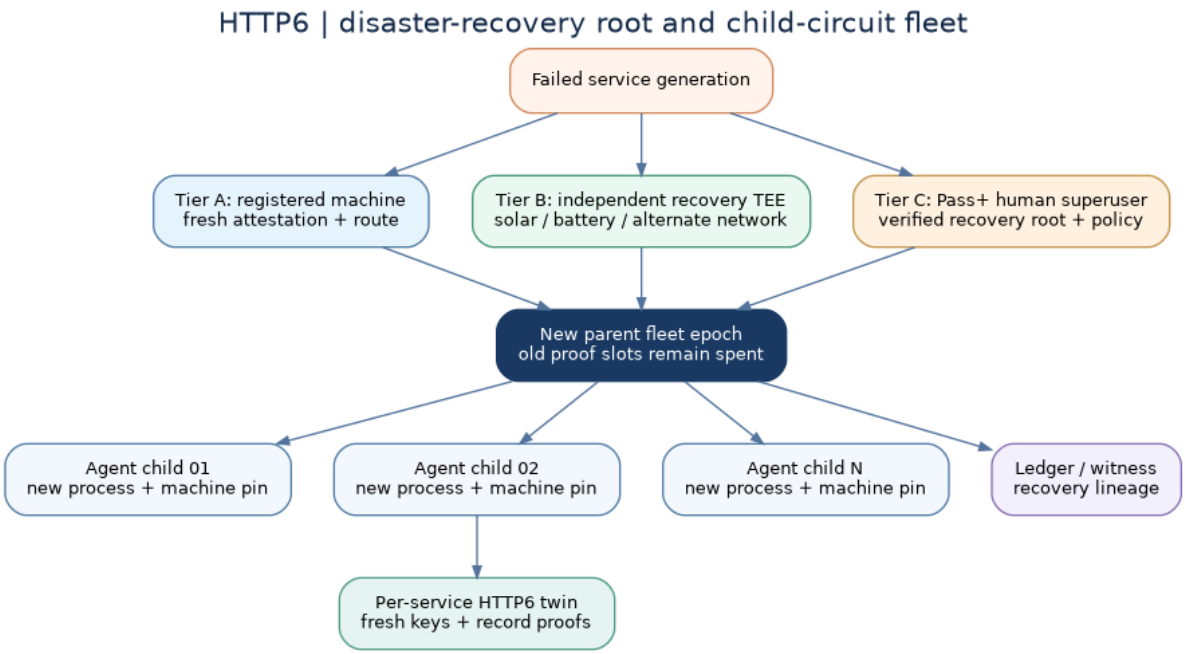

Figure 5: Three-tier recovery hierarchy: attested reboot, independent TEE and human recovery superuser, followed by fresh child generations.

```
  require_current_fleet_epoch: true
  approved_routes: [private_ingress, registered_service_endpoint]
independent_recovery:
  mode: pre_enrolled_tee
  power: autonomous_battery_solar
  alternate_network: authorized_disaster_link
  checkpoint: independently_retained
  issue_new_epoch: true
catastrophic_recovery:
  mode: human_superuser_ceremony
  recovery_root: pre_enrolled_offline_or_surviving_root
  create_new_parent_generation: true
  child_admission: machine_attestation_and_route_policy
  preserve_consumed_history: true
child_policy:
  per_machine_identity: required
  per_process_generation: required
  per_message_proof: required
  independent_revocation: true
```

This YAML is a source-level **illustrative policy schema**, not an already deployed service configuration. The deployment implementation must define signing, validation, authorization precedence, version migration, and storage semantics for each field. Any operator-side edit to fleet policy requires an authenticated administrative operation and an audit event.

II.14. Implementation work packages and direct deployment path

The program begins by specifying one authenticated request and one authenticated response through two synthetic, separately hosted executors. It then extends that primitive to a page with dependencies, a running browser session, cross-origin navigation, and a recoverable agent fleet. All packages share exact versioned protocol fixtures. The rollout deliberately establishes a small verifiable unit before expanding its execution and browser privileges.

WP-01 — Freeze the cryptographic and transcript specification. Select the supported KEM/key agreement, hash, HKDF, AEAD and MAC construction. Define canonical CBOR encoding, typed records, role and direction labels, key lifetime, rekey, protocol versions, authenticated service discovery, certificate or enrolled-attestation chain, and handshake transcript confirmation. Produce deterministic positive test vectors, corrupted-record vectors, and downgrade/reflection refusal vectors. The test must demonstrate that the client and server derive compatible directional secrets, and that different sequences, directions, and channel generations result in distinct record authentication contexts.

WP-02 — Human Admission Client to executor admission. Implement server availability and proof, trusted origin presentation, measured client process proposal, geometry challenge, consumed human decision, and twin process creation. Bind admitted runtime generation and service identity to the exact handshake. Verify a cloned executable, copied geometry response, forged server identity, changed route, and post-crash process all require fresh admission. Preserve saved personal circuits and existing Human Admission Client data during integration.

WP-03 — Wasmer WASIX HTTP6 service. Assemble a generic WASIX HTTP gateway with an admitted server circuit twin, resource budget, capability manifest, authenticated ProofCheck adapter, and durable ledger client. Deploy it to a controlled Wasmer Edge app using the current HTTP-server profile. Record the actual returned app URL, package identity, configuration and digest. Use a separate database or shared durable state service for nonce consumption and idempotency. Test instance stop/start, parallel app instances, invalid binary import, and wrong-tenant access.

WP-04 — HTTP6 browser workspace. Package the admission opening ceremony, channel adapter, HTML/content renderer, URL mapping, resource scheduler, manifest integrity checks, link interception, form submission, download handling and origin UI. Test a page containing styles, fonts, nested images, script-free interactions and link navigation. Introduce scripts only through a declared isolate that can neither bypass the HTTP6 scheduler nor directly invoke protected host operations. Run browser extension and native helper builds as distinct accepted products.

WP-05 — Agent executor and policy. Enroll one device, one measured WASIX executable and one exact service route. Start an authorized machine twin, verify every request and response, then reboot it. Exercise `attested_auto`, `human_on_reboot` and `break_glass_only` restart policies. Test wrong source address, attested-but-wrong executable, stolen process image, alternate IPv6 route and cloned binary.

WP-06 — Fleet admission and disaster recovery. Register a parent user, a modest test fleet, and one independently powered backup TEE. Qualify restoration from power loss, network isolation, alternate-network reentry, a failed primary checkpoint, and missing backup authority. Exercise human-superuser reconstitution against a known offline trust root, generate a fresh parent epoch and per-child admissions, and verify that a retired child cannot rejoin under its old proof chain.

WP-07 — Native transport. Implement native HTTP6 records on a controlled network with explicit local OS/browser integration. Match the `http6.n1` end-to-end confidentiality and integrity objectives independent of the outer HTTPS wrapper. Test active network interception, server impersonation, packet corruption, message replay, key compromise, forward-secrecy behavior and sustained load. Move one controlled asset/API service to a native HTTP6 endpoint while maintaining a separate conventional HTTPS service for incompatible clients.

WP-08 — Media and mesh. Begin with two endpoints and a relay; add three synchronized participants and a group membership epoch. Verify track provenance, per-packet or per-chunk proofs, out-of-order detection, slow consumer behavior, stream close, rekey, removal of one member and a reconnect requiring a new admitted generation. Publish latency and bandwidth measurements alongside successful secure behavior.

II.14.1 Milestones and acceptance conditions

Milestone	Implementation output	Required acceptance evidence
M0: source and protocol freeze	Exact codec, cryptographic suites, record profile and threat model	Byte-identical cross-language vectors; peer identity and downgrade refusal
M1: one-shot admission	Human admission proof instantiates one measured twin pair	Fresh geometry succeeds; replay and wrong-generation attempts fail
M2: encrypted oscillation	Request, response and heartbeat proofs on one live channel	Distinct proofs and keys; duplicated sequence consumed; crash retires generation
M3: WASIX web delivery	Gateway serves HTML, CSS, image and a second page	Each asset checked before rendering; resource and tenant isolation pass
M4: extension/native bridge	Customer browses in an HTTP6 workspace	Link and form navigation remain within the controlled scheduler
M5: agent/fleet	Attested machines, child delegation and reboot policies	Copied circuit and wrong route refused; revocation reaches descendants
M6: disaster recovery	Independent recovery TEE and human-superuser fallback	New epoch authorized, spent proofs retained, one fleet restart audited
M7: native channel	TLS-independent HTTP6 endpoint in a controlled environment	Active on-path and malicious endpoint tests; confidentiality/integrity acceptance
M8: scalable conference	Multi-recipient proof-bearing media	Authenticated track/packet identities, membership rekey, measured jitter

These are engineering completion gates and a build sequence. A source test, a deployed website, a proven cryptographic implication, and an accepted end-to-end channel are distinct forms of evidence. The release registry records each separately so product messaging identifies the exact accepted profile.

II.15. Operational security, monitoring, and recovery controls

A proof-based channel is operationally secure when the service can explain **who admitted it, which machine and process operated it, what resource it touched, what sequence was con-**

sumed, and which business effect completed. The HTTP6 observability schema therefore records opaque channel IDs, verified service identities, attestation decisions, parent/child lineage, policy epochs, sequence-advance summaries, route state transitions, and reconciliation references. It minimizes personal proof material, raw enrollment secrets, decrypted private circuits, traffic keys, and protected business payloads in general-purpose logs.

Operators monitor rejected handshake attempts, abnormal geometry failures, expired sequence positions, stalled heartbeats, route changes, failed attestation, rejected executable measurements, concurrent conflicting child starts, missing checkpoint members, rekey failures, and disaster epoch creation. An elevated rate of revoked proofs is a measurable symptom of replay attempts or client bugs; the remediation distinguishes those possibilities without exposing credential secrets in telemetry.

A controlled termination explicitly invalidates the generation and retires outstanding positions. A crash triggers remote suspicion and closes access after the heartbeat deadline. A recovered worker performs fresh admission, and business operations reconcile their original IDs. A human-controlled runtime reacquires an admission proof when strict policy requires it; a machine-controlled runtime uses an enrolled restart policy; a catastrophic fleet invokes one of the recovery tiers. Restored databases never treat a historical unused value as current without checking independently anchored consumption history.

Machine and route allowlists require ongoing updates as infrastructure changes. Vendor administrators enroll new IPv6 networks, planned proxy gateways, new WASIX release hashes, and changed hardware identity under governed policy. Changes produce a new policy epoch, which is recorded in the accepted channel and every new proof. An emergency proxy replacement cannot silently inherit a prior server identity simply because its hostname matches.

II.16. Open engineering decisions to fix in the protocol profile

A specification for first implementation must select a finite, reviewable parameter set: cryptographic suites and algorithm agility; exact peer-identity roots; hardware and TEE attestation format; proof-to-key-schedule binding; record codec; maximum message and chunk sizes; rekey thresholds; channel liveness timeouts; congestion, retransmission and ordering policy; persistent replay-state schema; acceptable clock skew; operating-system privilege model; and private-data retention. The same profile also fixes the exact route canonicalizer and policies for IPv4/IPv6 mobility. Selection is a product and engineering decision recorded in the release manifest rather than an untrusted on-the-wire default.

At the browser boundary, the initial renderer must decide which remote content types can execute code and how it enforces a content security policy. For the first milestone, an HTML/CSS/image/document subset with trusted link handling offers a measurable demonstration of secure web navigation. General-purpose JavaScript can follow after the renderer's network and permission interception is demonstrably complete. For a fully native browser, standards compatibility, accessibility, service workers, cache semantics, CORS/origin rules, and download protections deserve independent feature matrices.

At the recovery boundary, the design must specify custody of the offline root, last-accepted state preservation, approval quorum, authorized site list, how disaster epochs are propagated through isolated networks, the maximum child issuance permitted in an outage, and the exact test proving that retired authority cannot revive from a stale device image. An independently

powered laptop increases survivability only if its trust material, code, retained state and communications plan remain usable during the emergency.

II.17. Architectural conclusion

HTTP6 is proposed as a complete alternative web-channel architecture built around **human-admitted live circuits, independently authenticated service twins, per-message proof, synchronized encryption, WASIX-bounded execution, and policy-governed lifetime**. Its initial admission geometry ceremony gives a person the authority to open a particular measured runtime. Its oscillation gives that runtime a distinct cryptographic proof for each request, response, asset, media block and heartbeat. Its fleet model gives an administrator control over whole schools of agent runtimes while preserving each child's machine, route and operation identity.

A vendor can begin by exposing an HTTP6 gateway behind an existing web endpoint and gradually transfer resources to proof-enforced APIs. A controlled browser workspace makes the interaction visible. Native HTTP6 connections can later replace TLS on admitted channels by providing authenticated peer negotiation and encrypted record transport directly. The disaster-recovery hierarchy preserves operational continuity through machine-attested reboot, independent recovery TEE, and human-superuser reconstitution of a new fleet epoch. The resulting unit of trust is **a living, authenticated process making a specific request in the next verified channel state**, rather than a bearer artifact that remains useful merely because its expiration time lies in the future.

II. Appendix A. Vocabulary and source basis

HTTP6: proposed proof-native, proof-carrying web/API channel. **Human Admission Client:** user-facing admission and proof application. **HTTP6 reference architecture:** proof, custody, runtime, policy, authority and infrastructure services (Proof Authority, Enrollment Service, Minting and Delegation Service, Policy Decision Point, Authoritative State Plane, Consumption Ledger). **Rosario–Wang:** underlying private-reference proof construction. **Twin:** a pair of authenticated circuit executors maintaining synchronized channel state. **Oscillation:** continuing direction-specific exchange of authenticated, encrypted, fresh message records. **Human admission:** interactive local challenge required to create a specified process generation. **Child circuit:** a distinct policy-constrained machine or process issued under an authorized fleet parent. **Recovery TEE:** pre-enrolled protected recovery processor capable of creating a new recovery epoch under its assigned policy. **Disaster root:** surviving trust material and ceremony needed to establish fleet authority when regular recovery hardware is unavailable.

II. Public technical references

1. IETF, **RFC 9846**, *The Transport Layer Security (TLS) Protocol Version 1.3*, July 2026. https://www.rfc-editor.org/info/rfc9846/. Secure-channel requirements and engineering comparison.
2. IETF, **RFC 9334**, *Remote ATtestation procedureS (RATS) Architecture*, January 2023. https://www.rfc-editor.org/rfc/rfc9334.html. Attester, verifier, relying-party and freshness roles.
3. IRTF, **RFC 9180**, *Hybrid Public Key Encryption*, February 2022. https://www.rfc-editor.org/rfc/rfc9180.html. Example of a well-defined KEM/KDF/AEAD construction.

4. IETF, **RFC 9449**, *OAuth 2.0 Demonstrating Proof of Possession (DPoP)*, May 2023. https://www.rfc-editor.org/rfc/rfc9449.html. Comparison with existing sender-constrained request proofs.
5. Google Chrome for Developers, *Native messaging*. https://developer.chrome.com/docs/extensions/develop/concepts/native-messaging. Extension-to-native host boundary.
6. Google Chrome for Developers, *Declarative Net Request API*. https://developer.chrome.com/docs/extensions/reference/api/declarativeNetRequest/. Browser network interception and redirection constraints.
7. Google Chrome for Developers, *Deal with remote hosted code violations*. https://developer.chrome.com/docs/extensions/develop/migrate/remote-hosted-code. Packaged executable resources in Chrome extensions.
8. Wasmer, *Wasmer Edge Apps*. https://docs.wasmer.io/edge/learn/apps/. HTTP hosting and ephemeral application lifecycle.
9. NIST, *SP 800-34 Rev. 1: Contingency Planning Guide for Federal Information Systems*. https://csrc.nist.gov/pubs/sp/800/34/r1/upd1/final. Independent recovery, contingency and alternate facilities.

II. Document provenance

This is a **new architecture and engineering specification** prepared from the owner's description of the reference architecture and its Human Admission Client and supplied prior project materials. Public references document the surrounding transport, browser, attestation and recovery technologies. Internal source paths and implementation names in the source-layout sections are proposed module names, rather than claims that the completed HTTP6 protocol has been released. External vendors can adopt the specified profile in stages as each implementation milestone is accepted.

Part III. Executable Rosario–Wang attachment

The following mathematical and implementation appendix is reproduced from the supplied November 9 attachment. Its section labels and equation tags remain those of that source. Citation keys refer to the source register at the end of this part. Its original evidence statements retain their dates; the new reproduction is recorded in Part I and the regeneration evidence directory.

Abstract

This attachment converts the supplied finite Rosario–Wang formalizations into a runnable HTTP6-style demonstration. The code implements private enrollment, calibrated geometric membership, fresh additive masks, exact designated verification, authorized recovery, and an observer-visible record. New, explicitly identified engineering components bind that record to an executing instance, peer, application operation, inherited policy, and one-use event. A synthetic human admission activates an independently credentialed root; an authorized mint creates a distinct child; requests and responses carry fresh native membership proofs. Exact finite enumeration checks the released distributions and private decoding, while a symbolic resource calculation explains what the sequential privacy theorem means at one trillion observations. The executable reports its actual bounded tests separately from mathematical implications and production implementation obligations.

A. The implemented statement and private witness

The source uses commitment in the sense of an enrolled value and private dictionary. Define $X = (k, \varphi)$, where k is the enrolled string and φ is the private permutation of the response labels. The demonstrator retains this object in the designated private enrollment model. Its public identifier is an opaque instance or account handle, not an unsalted digest of a guessable secret. A separate cryptographic commitment scheme, with opening randomness and hiding/binding proofs, is not introduced by renaming that handle. [S1, §§1, 11–12; S2, §19.4]

For event e, the public statement x_e names the profile, complete event identity, nonce, frames, schedule, sender, intended verifier, channel, direction, stream, sequence, instance configuration, policy and lineage digests, operation identifier, exact synthetic payload, and expiration. The proof object contains this statement, the masked response word, and a separately keyed envelope authenticator. The actual private mask allocation is a protected setup resource, not a freely chosen existential variable. [S1, B1, B4; S2, E08]

$$w_t = \varphi(c_t(k_{r(t)})), \qquad z_t = (w_t + m_t) \bmod H. \tag{A1}$$

The cell function c_t identifies which public partition cell contains the selected character. The dictionary maps that cell to a private response. A fresh independent mask $m_t \in \mathbb{Z}_H$ is added before release. `raw_word`, `release`, and `respond` implement these operations. Calibration coordinates precede the data coordinates and use their own freshly allocated masks. This is the source primitive executed in each record, not a placeholder calling an unrelated proof library. [S1, Equations 1, 7, K2]

The designated verifier's native predicate is

$$\mathsf{NativeValid}_e \iff \big(\forall t:\ \varphi^{-1}((z_t - m_t) \bmod H) = c_t(k_{r(t)})\big) \wedge \mathsf{CalibrationValid}_e. \tag{A2}$$

`verify_native` checks the equivalent complete unmasked response word against the enrolled reference. A separate `recover` function takes only public profile information, the masked word, and actual masks. It reconstructs the dictionary from calibration and then the string from separating cell signatures. It never reads the reference store. That makes authorized recovery inspectable as a separate operation from comparison against enrollment. [S1, §§3–5; S2, T-03, T-05]

B. Geometry selection and the source profiles

The four-symbol ring implements the source's two offsets $(\nu, \nu+3) \bmod 4$ and its full two-response calibration. RW-12/4 implements the source's twelve distinct unequal coordinate pairs in $\mathbb{Z}_4^2$, nonce-indexed cyclic assignment, and three-response calibration. The 40-symbol profile implements the source's exact triple set

$$\mathcal{Q} = \{(a,b,c) : (c-a) \bmod 4 \in \{0,1\}\} \cup \{(a,b,c) : (c-a) \bmod 4 = 2,\ (b-a) \bmod 4 \in \{0,1\}\}. \tag{A3}$$

This set contains 40 distinct triples. Each coordinate gives a balanced four-cell facet, and the three coordinates separate all assigned symbols. The code fixes digits followed by uppercase letters as the 36 live symbols and uses four explicitly designated pads. A new nonce-dependent cyclic reindexing is an engineering extension: it permutes assignments of symbols to the source triples, so neither balance nor separation changes. The exact triple family and its separating argument are source-derived; the additional nonce adapter is identified as new. [S2, E44; S5, Lemma 16]

The public `challenge` command prints all cells for inspection. Private cell selections, dictionaries, and masking values are kept out of the observer export. This is a software presentation of the specified challenge interface; a protected display/input device is an implementation boundary to be supplied rather than simulated as a physical security result. [S1, §3; S3, A7]

C. Private entropy as a counted circuit resource

The finite tape is instantiated as a byte buffer. It is allocated once and consumed in disjoint bit positions. Each event receives a complete mask bundle before its challenge is released. Allocation advances a monotonic cursor, even if the event is later rejected, cancelled, or expires. Exhaustion stops issuance. Equal numerical mask values at different positions remain valid independent realizations in the ideal model; reusing the same random allocation is the excluded dependence. [S1, §§7–8]

For fixed cell count H and calibrated response count q, the exact independent-mask profile consumes

$$E_{\text{record}} = q \log_2 H, \qquad E_N = Nq \log_2 H. \tag{A4}$$

RW-12/4 uses $q = 2\ell + 3$. The implemented RW-40 profile uses $q = 3\ell + 3$. A 12-character keypad event therefore uses 54 ideal private bits. A 24-character machine event in RW-40 uses 150. At one trillion events those become 6.75 TB and 18.75 TB of continuously packed private mask material respectively. These quantities exclude provisioning, authentication keys, record metadata, durable state, and application data. One trillion request/response pairs use twice the per-record quantity. The calculation follows directly from the fixed encodings and counts, not a measured trillion-event run. [S1, E4, §10.3; S2, §14.1]

The runtime obtains provisioning bytes through the operating system's cryptographic random source. Exact enumeration separately ranges over the ideal finite mask distribution. These are two explicit evidence modes: arithmetic verification of the model and practical sampling for the executable demonstration. A production exact-privacy claim establishes the relevant private distribution and provisioning assumptions. Generating unlimited masks from a short seed instead establishes a computational profile, with its own analysis. [S1, E5; S2, §14.1]

D. The observer's posterior, including public code

Let B include the public program, permitted profile definitions, public instance records, fixed credential length, and admitted side information. For the native transcript, let G_e be the complete credential-independent public challenge and Z_e the combined calibration and data responses. Under the source's fresh private-mask model,

$$\Pr[Z_e = z \mid X = x, B, G_e = g] = H^{-q}. \tag{A5}$$

For each candidate credential, exactly one mask vector produces a given released word. Every vector has the same probability, so every candidate has the same observation likelihood. Public knowledge of the code does not change that count. The observer can compute many compatible explanations; those computations do not identify the historical mask or credential. The actual masks select the authorized private slice of the observation fiber. [S1, Equations 3–6; S3, Theorems 2 and 4]

The finite audit enumerates the complete RW-12/4 output law for all 288 one-symbol/dictionary pairs at a fixed nonce. Each pair produces all 1,024 possible five-coordinate outputs exactly once as its mask ranges over the 1,024 possibilities. It also checks the binary ring over all sixteen candidate nonces, two complete consecutive binary events, and calibrated RW-40 recovery. An exact `Fraction` calculation preserves a nonuniform prior, avoiding the mistaken inference that privacy forces every credential to become equally likely.

Sequential freshness gives

$$\Pr[\mathcal{T}_{1:N} = t_{1:N} \mid X = x, B] = \prod_{e=1}^{N} Q_e(g_e \mid B, t_{<e}) H^{-q_e}, \qquad I(X; \mathcal{T}_{1:N} \mid B) = 0. \tag{A6}$$

The public challenge kernels Q_e must be credential-independent given the prior public history, and each event must have the required fresh private masks. The product then contains no dependence on x; Bayes' rule preserves every supported prior. This source theorem supplies the

trillion-observation result. The program records that result as a **conditional mathematical consequence**, never as a trillion-logon benchmark. [S1, S1–S4; S2, E50]

The full runtime export additionally contains application metadata, real wall-clock fields, and computational tags. The exact finite audit fixes its public controls and does not establish secret-independent timing for Python or privacy for attacker-driven errors. The envelope's independent key and deterministic processing of the permitted transcript provide a compositional design path; a complete deployment analyzes all additional emissions jointly. If the acquired object includes the actual masks or enrolled secret, its observation model is different: the supplied recovery function demonstrates the resulting disclosure. [S1, S5–S6; S3, P5–P6]

E. Enrollment, origin, and application authentication

The native masked word does not by itself bind an arbitrary HTTP body. This attachment adds a **new engineering envelope**, separate from the source privacy theorem:

$$\tau_e = \mathsf{HMAC}_{a_v}(\mathsf{Enc}(\texttt{witness-envelope}, x_e, Z_e)) \,. \tag{A7}$$

The key a_v is provisioned independently of the human commitment and mask tape. The client checks a domain-separated challenge authenticator before applying its private material. The verifier compares the complete statement with the authoritative issued event and checks the independently supplied binding. That binding identifies sender, recipient, channel, direction, and stream. It is not accepted merely because an untrusted request writes those values in its own header.

Changing the target or payload changes the authenticated statement; using the record on a different bound channel fails comparison. The local component also checks the intended payload and admitted instance before responding. Tests cover these substitutions. Actual hardware-backed attestation, protected confirmation, and resistance to a physically relayed screen remain the deployment's trusted-context and interaction requirements. This executable checks their logical interface; it does not report a real hardware quote or a trusted-screen experiment. [S1, §§12.4–13; S6, action context and live-process authentication]

Envelope authenticity uses computational HMAC security. The source's zero credential information is a different proposition. The demonstration also leaves synthetic application payloads readable; protected production transport or reviewed authenticated encryption supplies confidentiality of arbitrary web contents. These distinctions allow an attachment to demonstrate a real native witness without overloading that witness with unsupported transport guarantees.

F. One-use validation and the local business effect

The verifier checks packet structure, exact stored context, live epoch, deadline, and envelope authenticity before claiming an event. Its first eligible complete response atomically transitions `ISSUED` to `BURNED`. It then performs native credential verification. A correctly authenticated envelope carrying wrong native responses consumes the event and is rejected. Repetition encounters `BURNED`; random invalid tags do not obtain the native-verification claim. [S1, B1–B4]

$$\mathsf{Accept}_e = \mathsf{ContextValid}_e \wedge \mathsf{Claim}_e \wedge \mathsf{CalibrationValid}_e \wedge \mathsf{DataValid}_e \wedge \mathsf{PolicyAllows}_e. \tag{A8}$$

The local SQLite implementation commits the burn before subsequent verification. Its effect table enforces a unique `(sender,operation_id)` pair and compares the canonical payload on reconciliation. A fresh valid request for the same business operation returns the previously recorded outcome; a changed payload under the same operation identity is rejected. The effect being established is a local table insertion. External transactions require an explicit idempotence and reconciliation contract with their own effect boundary. [S1, §13.3]

The test suite races multiple threads and two separate database connections for the same claim, reopens a consumed database, and retires unfinished events during restart. These are actual local observations. They are not a distributed consistency proof or protection against arbitrary rollback of the complete host.

G. Human origin and per-message machine proofs

The synthetic human's accepted ceremony activates a root machine instance. That root already has independent machine reference material. Its authorized `MINT` message creates a distinct child under a fixed approved template. The child receives neither the human credential nor a deterministic derivation of it. Every request subsequently uses the child's own calibrated membership relation and private tape. Every response uses the service's independent relation and private tape.

Effective permission remains an intersection along the registered ancestry:

$$\mathcal{A}_v = \bigcap_{u \in \mathrm{Anc}(v) \cup \{v\}} \mathcal{A}(P_u). \tag{A9}$$

The fixed reader template permits the approved report read and forbids export and further minting. The registry authenticates policy content and parent-grant linkage. A changed policy object fails its original grant check. Parent revocation is checked again after native verification and before effect creation. Root activation and minting are reached through admitted messages rather than through receipt of a public grant alone. The registry uses an independent issuer MAC in this attachment; a publicly verifiable deployment uses its specified digital-signature and trust-distribution profile. [S6, §§6.4–6.6]

This demonstrates the desired continuity: a human decision authorizes automation, while the automated messages carry fresh native proofs under distinct references. It also illustrates why privacy alone does not establish genealogy: the registry and policy gate supply the provenance and permission decisions in addition to the masking construction.

H. Efficiency and quantified experiments

For fixed alphabets, `respond` and `verify_native` evaluate a constant number of table operations per enrolled character. Calibration contributes constant overhead. `recover` uses a fixed separating-code inverse after privately recovering the dictionary. The total primitive work is $O(\ell)$ for these profiles. It does not search a candidate space or process all historical records. Full message handling additionally performs canonical encoding, HMAC, ancestry checks, and indexed event/effect operations.

The supplied benchmark measures 10,000 core records and reports actual elapsed time, Python version, and platform. It explicitly excludes the transport, envelope, ledger, hardware, and entropy-provisioning work. The test logs and JSON results supplied alongside this appendix are the authoritative measured values; no result is described as a throughput guarantee for the eventual protocol.

The source's distinct restricted blind-injection experiment gives

$$\Pr[\text{any acceptance in } Q \text{ eligible fresh attempts}] \leq \min\{1, QH^{-q}\}. \tag{A10}$$

That experiment gives the attacker no current masks, no current valid response, and no release oracle for the target event. The 24-symbol RW-40 profile has $q = 75$, so $H^{-q} = 2^{-150}$. With $Q = 10^{12}$ restricted attempts, the union bound is about 7.01×10^{-34}. This is not an unrestricted impersonation theorem or a bound on every MITM. It is printed with the precise access conditions that support it. [S1, A1]

I. Reproduction and evidence status

Run the demonstration, finite privacy audit, capacity calculation, and unittest suite using the commands in `README.md`. The package includes public sample records, a record schema, source hashes, a function-to-equation map, measured reports, and this Markdown attachment. The script is complete for the stated local experiment and has no undefined primitive in its forward response, designated verification, or private extraction path.

The executable additions are new reference code authored for this review task. The mathematical source establishes the conditional claims; the Python tests establish the reported finite behavior. The supplied Lean registry provides exact declaration names but explicitly distinguishes source correspondence from a new kernel run. The separate exact-projection paper is not used to claim polynomial SAT solving or computational hardness of passive credential recovery. This attachment needs neither conclusion. [S4; S5]

Source register

[S1] `Rosario_Wang_ENI6MA_Consolidated_Formalization.md`, 16 September 2026. Primitive, ring/keypad construction, calibrated recovery, nonce/mask entropy, event burn, sequential privacy, and restricted active game.

[S2] `ROSARIO-WANG-COMPLETE-FORMALIZATION(2).md`, 14 September 2026. Finite incidence encoding, private reference, calibration extension, exact 40-symbol facets, and information/resource accounts. The separate unnumbered attachment could not be read or recovered in this run; no byte identity is assumed.

[S3] `ENI6MA-Unified-Geometric-Formalization.md`, 13 September 2026. Candidate intersections, public fibers/private slices, conditional privacy, and additional-observation analysis.

[S4] `B-lean-formalization-registry(1).md`. Theorem/source correspondence, including modular release, privacy, separation, joint extraction, and the stated evidence boundary.

[S5] `ROSARIO-WANG-EXACT-PROJECTION-FULL-FORMALIZATION.md`, version 1.1, 14 September 2026. Exact three-facet chamber construction in §17; separate polynomial-compression obligation is not used by this attachment.

[S6] `ENI6MA_White_Paper_Human_Originated_Identity.md`, 8 November 2026. Prior supplied architecture for action-bound MACs, synchronization, inherited policies, and resource checks. The implementation choices here remain explicitly identified engineering glue.

[E1] Python official `secrets` documentation: https://docs.python.org/3/library/secrets.html

[E2] Python official `hmac` documentation: https://docs.python.org/3/library/hmac.html

[E3] SQLite official transaction documentation: https://www.sqlite.org/lang_transaction.html

Appendix A. Complete executable-envelope schema

This is the unchanged JSON Schema for the delivered local review envelope. Its representation constraints are supplemented by the canonical codec, exact stored context, designated masks, current authority, and ledger checks. The schema does not itself supply an authenticator or define the final N1 wire encoding.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:http6:review:rw-envelope:1",
  "title": "HTTP6 Rosario-Wang review envelope (not an adopted HTTP6 wire format)",
  "description": "Representation validation only. Stored context, H-dependent residue ranges,
  ↪  canonical bytes, policy, masks, state, and authenticity are checked separately.",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "statement",
    "released",
    "proof_tag"
  ],
  "properties": {
    "statement": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "epoch",
        "event",
        "sender",
        "recipient",
        "channel",
        "direction",
        "stream",
        "device",
        "measurement",
        "operation_id",
        "protocol",
        "kind",
        "sequence",
        "nonce",
        "issued_ns",
        "expires_ns",
        "policy_digest",
        "lineage_digest",
        "request_ref",
        "payload",
        "profile",
        "length",
        "calibration_cells",
        "responses",
        "frames"
      ],
      "properties": {
        "epoch": {
          "type": "string",
```

```
          "minLength": 1
        },
        "event": {
          "type": "string",
          "minLength": 1
        },
        "sender": {
          "type": "string",
          "minLength": 1
        },
        "recipient": {
          "type": "string",
          "minLength": 1
        },
        "channel": {
          "type": "string",
          "minLength": 1
        },
        "direction": {
          "type": "string",
          "minLength": 1
        },
        "stream": {
          "type": "string",
          "minLength": 1
        },
        "device": {
          "type": "string",
          "minLength": 1
        },
        "measurement": {
          "type": "string",
          "minLength": 1
        },
        "operation_id": {
          "type": "string",
          "minLength": 1
        },
        "protocol": {
          "const": "http6-rw-review-1.0"
        },
        "kind": {
          "enum": [
            "ADMISSION",
            "MINT",
            "REQUEST",
            "RESPONSE"
          ]
        },
        "sequence": {
          "type": "integer",
          "minimum": 1
        },
        "nonce": {
          "type": "integer",
```

```
          "minimum": 0,
          "maximum": 340282366920938463463374607431768211455
        },
        "issued_ns": {
          "type": "integer",
          "minimum": 0
        },
        "expires_ns": {
          "type": "integer",
          "minimum": 0
        },
        "policy_digest": {
          "type": "string",
          "pattern": "^[0-9a-f]{64}$"
        },
        "lineage_digest": {
          "type": "string",
          "pattern": "^[0-9a-f]{64}$"
        },
        "request_ref": {
          "type": [
            "string",
            "null"
          ]
        },
        "payload": {
          "type": "object",
          "required": [
            "operation",
            "resource",
            "destination"
          ],
          "properties": {
            "operation": {
              "type": "string",
              "minLength": 1
            },
            "resource": {
              "type": "string",
              "minLength": 1
            },
            "destination": {
              "type": "string",
              "minLength": 1
            }
          },
          "additionalProperties": true
        },
        "profile": {
          "enum": [
            "ring4",
            "rw12",
            "rw40"
          ]
        },
```

```
        "length": {
          "type": "integer",
          "minimum": 1,
          "maximum": 1024
        },
        "calibration_cells": {
          "type": "array",
          "items": {
            "type": "integer",
            "minimum": 0,
            "maximum": 3
          },
          "minItems": 2,
          "maxItems": 3
        },
        "responses": {
          "type": "integer",
          "minimum": 4,
          "maximum": 3075
        },
        "frames": {
          "type": "array",
          "minItems": 2,
          "maxItems": 3,
          "items": {
            "type": "array",
            "minItems": 2,
            "maxItems": 4,
            "items": {
              "type": "string",
              "minLength": 1
            }
          }
        }
      }
    },
    "released": {
      "type": "array",
      "minItems": 4,
      "maxItems": 3075,
      "items": {
        "type": "integer",
        "minimum": 0,
        "maximum": 3
      }
    },
    "proof_tag": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}$"
    }
  }
}
```

Appendix B. Payload-case acceptance register

Each scenario generates fresh synthetic enrollments, contexts, masks, and proofs. Expected failures are successful tests when the exact required rejection and effect count occur. Complete input JSON and newly observed artifacts are in the validation and evidence directories.

Case	Scenario	Expected decision	Local effects
P01	Fresh request and separately proven response	`ACCEPTED`	1
P02	Body substitution invalidates the original statement	`REJECTED_EXCEPTION`	0
P03	Destination substitution invalidates the original statement	`REJECTED_EXCEPTION`	0
P04	Repeated consumed record creates no second effect	`REPLAY`	1
P05	Authenticated export still fails inherited policy	`REJECTED`	0
P06	Verifier rejects another channel	`REJECTED_EXCEPTION`	0
P07	Local participant rejects a relayed context	`REJECTED_EXCEPTION`	0
P08	Wrong proof direction is rejected	`REJECTED_EXCEPTION`	0
P09	Invalid envelope tag does not consume an eligible native attempt	`REJECTED_EXCEPTION`	0
P10	Wrong native word burns the authenticated attempt	`REJECTED`	0
P11	Retired parent fences a prepared valid proof	`REJECTED`	0
P12	Changed child policy fails original admission	`REJECTED`	0
P13	Leaf circuit cannot create an authorized grandchild	`REJECTED_EXCEPTION`	0
P14	Fresh proof for same business operation reconciles	`RECONCILED`	1
P15	Same business identifier cannot conceal changed content	`OPERATION_CONFLICT`	1
P16	Response naming an unrelated recipient fails request linking	`REJECTED`	1

Appendix C. Schema, model, and source-code inventory

Asset group	Complete supplied files
architecture	`Authority_Policy_Schema.json`; `protocol_contracts.schema.json`; `review_objects.schema.json`
assurance examples	`Authority_Policy_Example.json`; `Authority_Policy_Example.yaml`; `additional_classification_examples.json`; `canonical_test_vector.json`; `csf_outcome_examples.json`; `cui_family_examples.json`; `family_examples.json`; `review_action_proof.json`; `review_child_grant.json`; `review_evidence_record.json`; `review_policy.json`; `review_public_registry.json`; `review_request.json`; `review_root_grant.json`
models	`architecture.mmd`; `state_machine.yaml`
review runtime	`http6_review_envelope.schema.json`; `validation_suite.schema.json`
test payloads	`native_vectors.json`; `validation_cases.json`

The full contents of each listed file are supplied in the ZIP. The generic policy and protocol-contract schemas are architecture/reference artifacts distinct from the executable envelope schema. The expanded review volume also embeds its source code, schemas, and example catalog as appendices. Reproduction instructions remain inside each preserved standalone package.

Appendix D. Submission and authorship controls

The owner-supplied author and advisor credits are carried forward in the current submission front matter and revised IEEE manuscript. The original formalizations, architecture, and evidence papers retain their original identities. No affiliation, ORCID, assent, signature, funding statement, or correspondence address has been invented. Before sending, the authorized submitter confirms contact and affiliation information, coauthor/advisor approvals, the public or restricted disclosure scope, and any applicable contribution terms.

The cover letter requests technical review and a suitable demonstration path. It does not represent an accepted NCCoE collaboration, a CMVP application, a registered HTTP version, a publication acceptance, or an executed transmission. The native-profile decision register and the NIST requirement/evidence assignments accompany the request so that the receiving team can see both the available evidence and the work it is being asked to discuss.