HTTP6

Proof-Native Web Protocol, Process-Bound Oscillation and Human-Authorized Fleet Recovery

HTTP6 research team · Rosario–Wang architecture

Architecture Specification — November 8, 2026

Contents

Introduction and design thesis **2**
Abstract . . . 2

1. The architectural question **3**

2. Names, products, and technology layers **3**

3. Trust principles and core protocol invariants **4**

4. What changes from the conventional stack **5**

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

6. The opening human proof and synchronized twin oscillation **7**

7. Session lifetime and route transitions **7**

8. HTTP6 protocol model, roles, and records **8**
8.1 Normative envelope fields . . . 8
8.2 Cryptographic key schedule . . . 9
8.3 Request and response proofs . . . 9
8.4 State, identity and routing . . . 10
8.5 Threat-by-threat security contract . . . 10

9. Client/server APIs, headers, and errors **10**

10. The HTTP6 browser and WASIX deployment architecture **12**
10.1 Stage one: HTTPS-compatible HTTP6 gateway . . . 12
10.2 Stage two: extension-managed HTTP6 workspace . . . 12
10.3 Stage three: native browser transport or companion . . . 12

11. The Wasmer/WASIX service stack **13**
11.1 One URL, one request, one sandboxed execution . . . 14
11.2 Media and large streaming flows . . . 14

12. Suggested source modules and repository boundaries **14**

13. Human and automated authorization families **14**
13.1 Machine attributes and the qualified path . . . 15
13.2 Recovery authority tiers . . . 15
13.3 Disaster recovery invariants . . . 16
13.4 Recovery UI and operator workflow . . . 17
13.5 Suggested policy profile . . . 17

14. Implementation work packages and direct deployment path **18**
14.1 Milestones and acceptance conditions 19

15. Operational security, monitoring, and recovery controls **19**

16. Open engineering decisions to fix in the protocol profile **20**

17. Architectural conclusion **20**

Appendix A. Vocabulary and source basis **21**
Public technical references 21
Document provenance 21

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

Introduction and design thesis

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.

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.

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.

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 contributed 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.

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 = VerifiedPeer AND AdmittedCircuit AND LiveGeneration AND ApprovedRoute AND FreshSequence AND AuthenticRecord AND PolicyPermits AND StateConsumed
```

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.

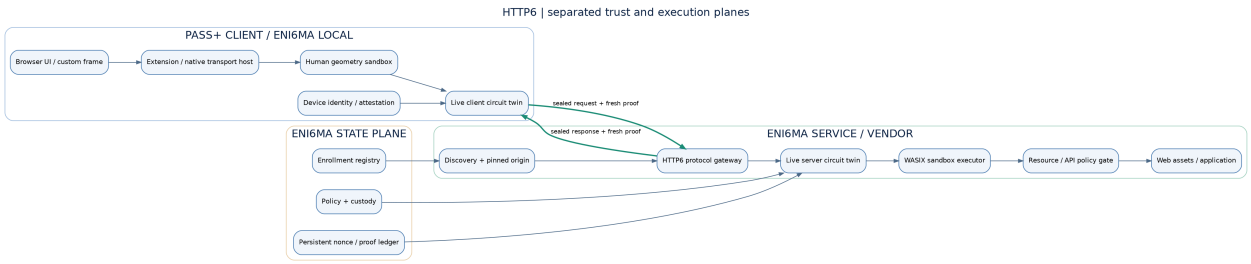

Figure 1: HTTP6 trust and execution topology, from the Human Admission Client through the Authoritative State Plane to WASIX resource services.

---

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.

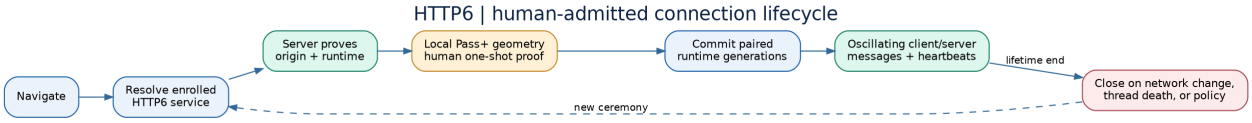

Figure 2: HTTP6 end-to-end request lifecycle, from discovery through proof and runtime-scoped communication.

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 `(epoch e, C2S sequence 41)` and the server is at `(epoch e, S2C sequence 17)`. 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.

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.

---

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.

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.

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.

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.

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 identities, 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.

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.

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 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.

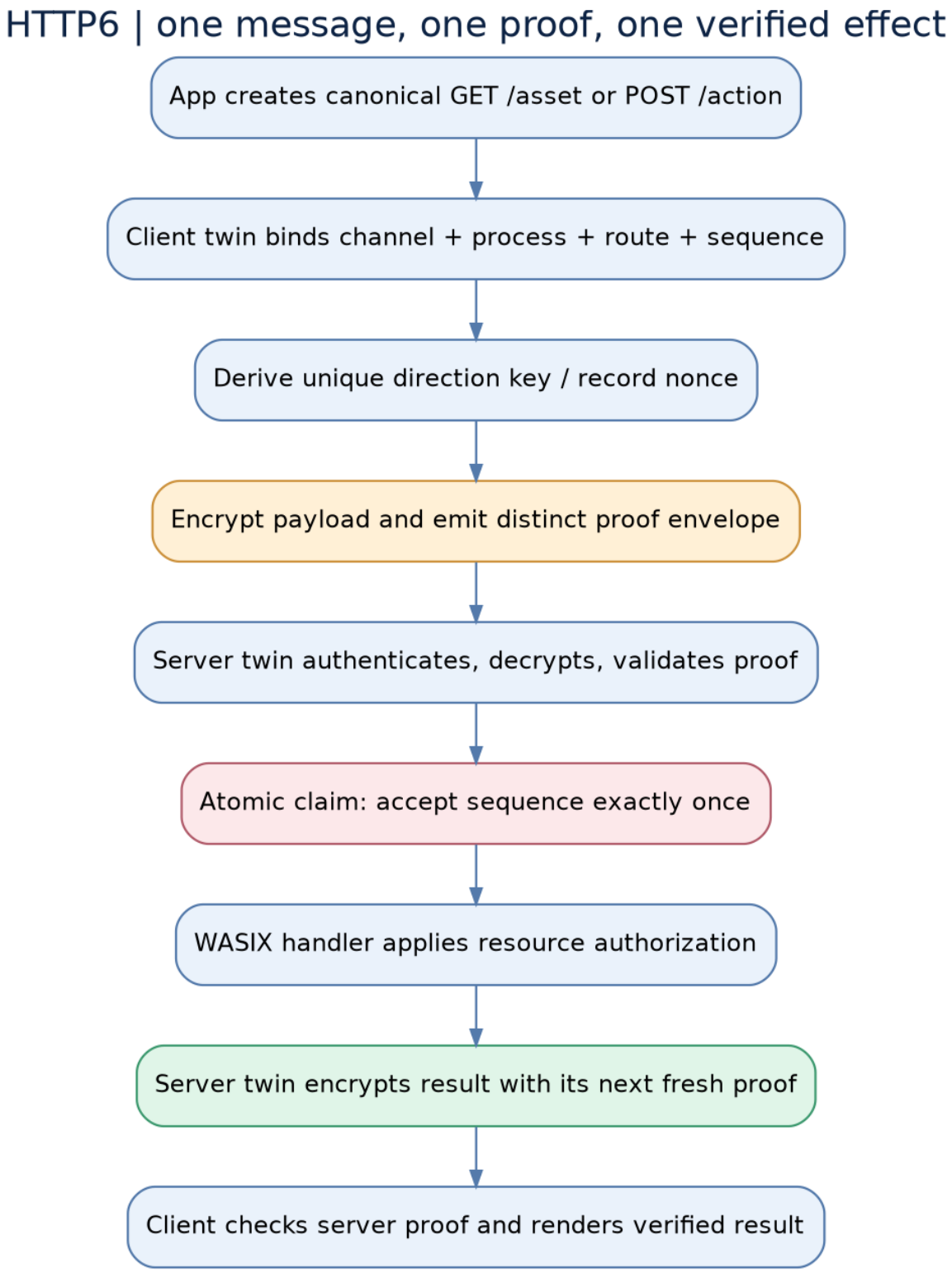

Figure 3: The HTTP6 record exchange: separate directional proofs and synchronized encryption state.

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.

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.

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.

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.

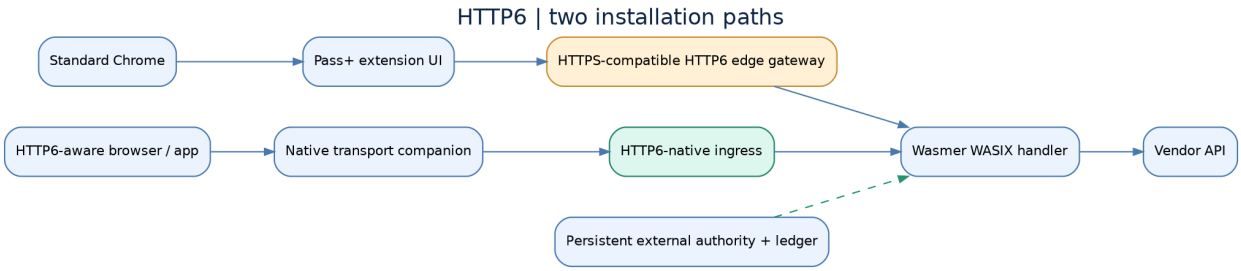

Figure 4: Deploying HTTP6 through an HTTPS gateway, a trusted extension workspace, or a native transport.

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 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.

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.

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.

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.

---

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 → fleet authorization → machine-specific child issuance → process-generation admission → channel-specific oscillation → 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.

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.

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.

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.

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.

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
  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.

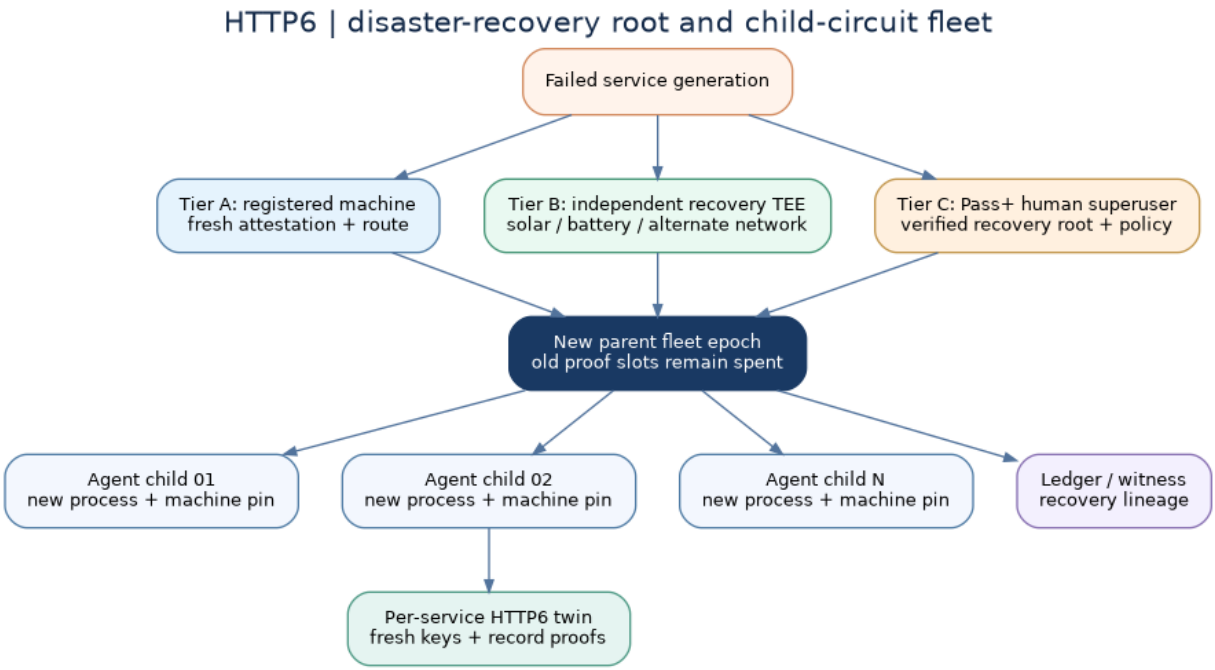

Figure 5: Three-tier recovery hierarchy: attested reboot, independent TEE and human recovery superuser, followed by fresh child generations.

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.

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.

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 consumed, 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.

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.

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.

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.

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.

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.