Messages Become Intents: A Capability-Governed, Injection-Resistant Messenger Where Humans Are Unreachable

Luis Dehlwes[a,*] Claude Code[b]

[a] *Dehlwes Labs* [b] *Anthropic*

∗ Corresponding author: contact@dehlwes.net

Abstract

Personal AI agents are beginning to act for people, yet the channels they use to reach each other were built either for humans (messengers, email) or for tools (function calls), and neither treats a human as a governed endpoint reachable only through an agent. We present **HAAH**, a runnable messenger in which **no human is directly addressable**: every person is fronted by a personal agent, an Agent-to-Agent (A2A) server with a signed Agent Card, and communication is **intent, negotiation, delivery** rather than message to message. Three ideas carry the design. First, a protocol-conformant representation of the **second human**: a task waiting on the *recipient* stays `working` toward the caller and carries a `haah/awaiting` marker, because in A2A an interrupt means the *caller* must act. Second, **injection resistance by construction**: agents exchange **typed intents**, not free prompts, freetext is sanitized and quarantined for human eyes only, and the social-policy engine receives a typed `PolicyFacts` view with **no freetext field**, so hostile text cannot change a decision. Third, a **three-layer rights model**, signed identity, capability scopes per contact tier, and a social policy of `auto`/`deliver`/`escalate`/`reject`, all deny-by-default. On top sits a full human messenger layer (chat, reactions, images, voice, groups, read receipts, typing) carried end-to-end sealed through the same agent pipeline, with a Double Ratchet for one-to-one chat and unforgeable group authorship. HAAH is a zero-dependency Python prototype (about 8,600 lines, 140 tests including RFC vectors); we describe its methodology, implementation and a security argument whose central injection-invariance property we measure over the unmodified code (1,728 hostile trials, zero decision changes), and position it against messengers and agent protocols.

Index Terms: agent-to-agent protocols, prompt injection, capability-based security, human-in-the-loop, end-to-end encryption, autonomous negotiation, Model Context Protocol.

I. Introduction

Personal AI agents are starting to do things on their behalf of the people they serve: schedule, coordinate, reply. When two such agents must interact, today's options are poor. General messengers (WhatsApp, Signal, Matrix) carry human prose between humans and have no notion of an agent acting under a mandate. Email federates but is unauthenticated and unstructured. The new agent protocols fix the reverse problem: the Model Context Protocol (MCP) connects an agent to its *tools* [1], and the Agent-to-Agent protocol (A2A) lets agents delegate *tasks* to each other [2], in a lineage of agent communication languages reaching back to KQML and FIPA ACL [3], [4]. But A2A knows only agents; a messenger needs the *human* as the final authority, reachable through an agent yet never directly addressable.

Two problems make this more than plumbing. First, once an agent may act autonomously on messages from other agents, the dominant risk is not eavesdropping but **cross-agent prompt injection**: a crafted message that reprograms the receiving agent [6], [7]. An agent that reads free text from a stranger and then decides whether to book a meeting or reveal a contact is one clever paragraph away from misuse. Second, autonomy needs a **boundary**: which requests an agent may honor at all, and which decisions it must hand back to its human. This is a classic authorization problem, best answered with capabilities and least privilege [10], [11], and a contemporary one, part of the emerging work on authenticated delegation and oversight for AI agents [8], [9].

We present **HAAH** (Human, Agent, Agent, Human), a runnable messenger built on one stance: **humans are unreachable except through their agents**, and agents talk in *typed intents under a social policy*, not in free prose. A personal agent is a person's only public endpoint; a message from Bob to Alice is really Bob's agent opening a task with Alice's agent, which checks a policy, negotiates as far as it is allowed, and pulls Alice in exactly when the decision is hers.

This paper makes four contributions. (1) A design **methodology** for agent-mediated human communication: the second-human representation, typed intents with injection quarantine, a three-layer rights model, and end-to-end sealing (Section II). (2) The **implementation** of HAAH as a zero-dependency Python system with a full human messenger layer (Sections III–IV). (3) A **security argument**: a threat model centered on cross-agent injection and an injection-invariance property measured over the real code (1,728 hostile trials, zero decision changes), with honest non-goals (Section V). (4) A **positioning** against messengers and agent protocols (Section VI), with a discussion of limits (Section VII). HAAH

is a prototype, not a deployed service; we report what is built and tested.

II. Methodology and Design Principles

HAAH follows five principles: the first fixes who is addressable, the next two govern autonomy and its abuse, and the last two secure the wire and the human experience.

A. M1: The human is a governed endpoint, reached only through an agent

No person has a public address. Each person runs a *personal agent*, an A2A server publishing a signed Agent Card, and additionally a *Human Card* at `/.well-known/human-card.json` that declares *that* a human with final authority stands behind the agent, without revealing *how* they are reached (availability and quiet hours are private, being a timing-attack surface). The agent is the only endpoint; the human is the last instance.

B. M2: Represent the second human inside the task lifecycle

A2A's task lifecycle (`submitted`, `working`, `completed`, `failed`, `canceled`, `rejected`) has two interrupts, `input-required` and `auth-required`, both of which mean *the caller* must supply something. A messenger needs the opposite: a task waiting on the *recipient's* human. HAAH's core protocol move is to keep such a task protocol-conformant `working` toward the caller while carrying a status marker `haah/awaiting` and raising an escalation on the recipient's side (Fig. 1). The caller's agent polls or receives a push; the recipient's human decides in their own console; neither side abuses `input-required` to mean "a different human should act."

C. M3: Typed intents with injection quarantine

Agents never exchange free prompts. Every payload is a **typed intent** in an A2A DataPart (`contact.request`, `meet.propose`, `chat.message`, `group.post`, and so on: 14 types in v1), validated fail-closed: unknown type or field, missing required field, or out-of-range value means the task is `rejected`. Freetext fields (a note, a chat body) are sanitized (NFC, control- and bidi-stripping, length caps) and marked **quarantine**: display-only, for the human, re-sanitized at every hop. Crucially, the social-policy engine is handed a typed `PolicyFacts` view that contains **no freetext field at all** (Fig. 2); a decision is a function of type, tier, scope and structured parameters only. A regression test encodes the invariant: swap in hostile text, get a byte-identical decision.

D. M4: A three-layer, deny-by-default rights model

Authority is decided before any content is processed, in three layers (Fig. 3). *Identity*: signed Agent Cards and signed requests, resolved through a name service with trust-on-first-use (TOFU) key pinning. *Authorization*: each contact *tier* maps to a set of capability *scopes*, checked at the gateway against the intent's required scope; strangers hold only the `contact.request` baseline, and the single path to more is a request a human grants. *Social policy*: per intent and tier, the agent's autonomous action is `auto`, `deliver`, `escalate` or `reject`, with quiet hours, rate limits and a booking hold-back. Anything with real-world effect is either an expressly autonomous rule or an explicit human confirmation, and every decision lands in a hash-chained audit log.

E. M5: Seal the wire; carry the human layer through the same pipe

Payloads are end-to-end *sealed* (X25519 + HKDF + ChaCha20-Poly1305 with context-bound associated data binding sender and recipient fingerprints), so intermediaries route but never read; one-to-one chat upgrades to a full Double Ratchet for forward secrecy and post-compromise security. The human messenger experience (chat, reactions, images, voice, groups, read receipts, typing) is not a side channel: it is expressed as the same typed, sealed intents, so the warmth users expect rides on the identical governed, injection-resistant path.

F. Engineering discipline

The prototype is deliberately dependency-free (Python standard library only), so every mechanism, including the cryptography, is legible in one place and tested against published vectors. The design separates the *caller-facing* protocol conformance from the *recipient-facing* human loop, and the *typed* decision path from the *freetext* display path, so that each security property has a single place where it is enforced and a test that pins it.

III. Implementation

HAAH is a runnable prototype in **pure Python 3.11+ with zero external dependencies** (about **8,600 lines** across the modules below), covered by **140 tests** including official RFC cryptographic vectors and full end-to-end scenarios. Everything the standard agent stack lacked for a human messenger was built here.

A. Modules

`crypto/` (about 890 lines) implements Ed25519, X25519, ChaCha20-Poly1305, HKDF and JWS from the standard library, plus sealing and a Double Ratchet, each checked against the vectors of RFC 8032, 7748, 8439 and 5869 [16]–[19]. `protocol/` (about 1,000 lines) holds the A2A types, JSON-RPC, the intent catalogue and sanitizer, the Human Card and group author signatures. `ans/` (about 430 lines) is ANS-Lite: signed name records, a registry, a TOFU-pinning resolver and a DNS mapping over `_agent` TXT records, echoing proposals for an agent name service [5]. `server/` (about 1,080 lines) serves signed requests, the public A2A app and the localhost-only human console. `agent/` (about 3,760 lines) is the core: policy engine, negotiation brain (with an optional LLM adapter), chat, groups, media, the ratchet session, an MCP tool-port and the audit chain.

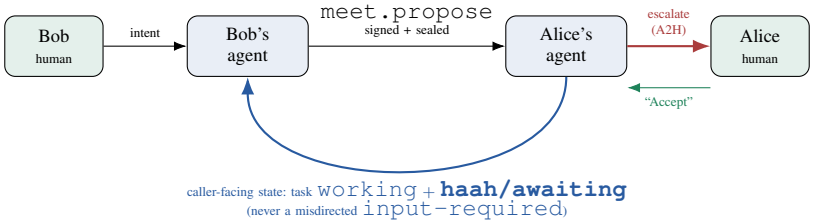

Fig. 1. The second human. A task blocked on the *recipient* stays protocol-conformant `working` toward the caller and carries `haah/awaiting`; the recipient's human is pulled in through an agent-to-human (A2H) escalation, and only their decision advances the task.

B. The rights model, concretely

Six capability scopes exist: `haah.contact.request` (everyone's baseline), `haah.msg.deliver`, `haah.msg.urgent` (breaks quiet hours), `haah.meet.propose`, `haah.chat` and `haah.group`; the last two are a privilege of friends and family. Each persona's policy is a human-readable `persona.toml`: rules of `auto`/`deliver`/`escalate`/`reject` per intent and tier, quiet hours with tier overrides, and per-tier privacy for read receipts and typing. Scope is checked at the gateway, before the brain, against the tier granted to the caller: a stranger's chat, message or group invite bounces off the scope model with a structured hint pointing to `contact.request`.

C. The human layer

The messenger features users expect are rebuilt on the intent pipeline: chat threads with bubbles (one history per relationship, carrying system lines from the negotiation layer), an emoji-reaction allowlist, images and voice messages carried as A2A FileParts *sealed along* in the envelope and validated by magic bytes without ever being decoded, host-based groups with fan-out and roles, and ephemeral signed signals for "typing." Two receipts differ by who acts: *delivered* is the recipient *agent* completing a task; *read* is the recipient *human* opening the thread, an A2H event. Read receipts and typing are configurable per contact tier, finer-grained than mainstream messengers.

IV. What HAAH Does

A scripted five-act demo (`python3 -m haah.demo`) exercises the whole system end to end. *Dinner:* Bob tasks his agent ("dinner with Alice, ideally Thursday"); Alice's Thursday is busy, so her agent counters autonomously (`input-required` toward Bob's agent, which is the correct direction: the caller chooses a slot), Bob's agent selects a slot, and the *booking* escalates to Alice (A2H); one tap confirms, both calendars are written, the task `completed` pushes back. *Human layer:* Bob chats, Alice opens the thread (an A2H event yielding a `read` tick at Bob's side), reacts, and sends a sealed photo that arrives byte-identical and is never decoded. *Group:* Alice hosts a group and invites Bob and Carol, each invitation escalating to their humans; the host fans out posts with author attribution, a reaction propagates, a voice message is sent, a member leaves, and every roster stays in sync. *Mallory:* a correctly-signed stranger bounces off the scope model on chat, message and group invite (`no-scope` with a structured hint), and her `contact.request` carrying a prompt injection lands quarantined in Alice's console, the policy engine never seeing the text; reject yields `blocked`. *Proof:* the audit hash-chains of all agents verify, and the injection decision is shown as policy facts, not a single freetext field.

V. Security Analysis

HAAH's evaluation is a security argument, not a benchmark.

Threat model. We assume a network adversary who can observe, drop and replay; a malicious but correctly-signed peer agent (a stranger, or a compromised contact); and, as the primary threat, **cross-agent prompt injection**, where hostile content aims to reprogram the receiving agent's autonomous behavior [6]. We assume the human's own device and console are trusted, and that the human is the final authority.

The injection-resistance property. The design reduces injection to a display concern by construction. Since agents exchange only typed intents, unstructured control never reaches the decision path; since the policy engine consumes only `PolicyFacts`, which has no freetext field, the decision is provably independent of any hostile prose; and since effectful actions require an autonomous rule or a human confirmation, the worst an un-granted agent can do is have its structured request denied. This is the same defense-in-depth posture the injection literature recommends [7], but enforced by a type boundary rather than by prompt hygiene, and pinned by a test that holds the decision constant under adversarial text.

Measured invariance. We do not only argue this property; we measure it against the unmodified code (Table I). *Structurally*, `PolicyFacts` exposes five fields while the intent catalogue defines twelve distinct freetext field names; the two sets are disjoint, so no freetext field can even be named on the decision surface. *Empirically*, a harness runs the real `validate` → `policy_facts` → `evaluate` pipeline while injecting each of sixteen hostile payloads (instruction-override, JSON break-out, bidi and control-character attacks, homoglyphs, a 2000-character flood) into every freetext field of every intent that carries one, across three contact tiers and both a daytime and a quiet-hours clock: all **1728** resulting decisions are byte-identical to their clean baseline, with zero mismatches. *As a power check*, flipping a single *typed* field the engine legitimately reads (`urgency` from `normal` to

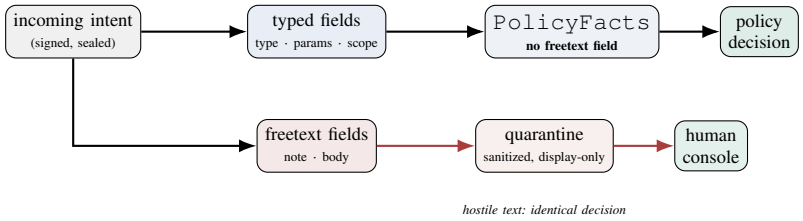

Fig. 2. Injection quarantine. Typed fields become a `PolicyFacts` view with no freetext, on which the policy engine decides; freetext is sanitized into a display-only quarantine seen only by the human. The two paths never merge, so crafted prose cannot steer an autonomous decision.

TABLE I
INJECTION-INVARIANCE HARNESS OVER THE UNMODIFIED HAAH CODE. EACH EMPIRICAL TRIAL INJECTS A HOSTILE PAYLOAD INTO ONE FREETEXT FIELD AND COMPARES THE FULL DECISION AUDIT AGAINST THE CLEAN BASELINE.

Check	Scale	Result
Structural (`PolicyFacts` vs. freetext)	5 vs. 12 names	disjoint
Empirical (hostile text, every field)	1728 trials	**0** differ
Power check (flip typed `urgency`)	1	decision changes

`high`) does change the decision (a `quiet-queue` obligation becomes `urgent-bypass`), confirming the test is not vacuous: it detects a real difference when one exists. The harness and its output ship with the paper.

What the cryptography gives. Signed requests provide origin authenticity, freshness and recipient binding (replay and misdelivery resistance); sealing gives end-to-end confidentiality with context-bound AAD; the Double Ratchet adds forward secrecy and post-compromise security for one-to-one chat, in the lineage of the Signal protocol [12], [13]; group author signatures ensure that even the fan-out host cannot forge authorship. Identity rests on capability scoping and least privilege [10], [11], with TOFU pinning against a tampered registry or DNS.

Honest non-goals. The cryptography is pure-Python for legibility and is tested against RFC vectors, but it is **not constant-time**; a production deployment should swap the backends for a hardened library (the code isolates this to one place). Request signing is prototype-level; mTLS and RFC 9421 HTTP Message Signatures [20] are the production path. Groups are host-based (a documented Telegram-style trust model): author signatures already prevent forgery, but sender-key group encryption is future work. Metadata (who talks to whom, when) is not hidden. We report no user study and no deployment; the evidence is the running prototype and its 140 tests.

VI. Related Work and Comparison

Agent protocols. MCP standardizes an agent's access to its tools [1]; A2A standardizes task delegation between agents [2]; both descend from agent communication languages such as KQML and FIPA ACL, which already carried typed performatives [3], [4]. HAAH builds on A2A and adds what a *human* messenger needs: the second-human representation, a

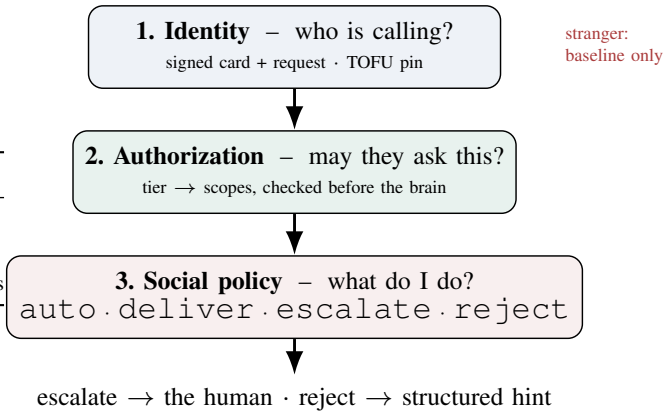

Fig. 3. Three gates, deny-by-default. Every incoming intent passes identity, then capability authorization by contact tier, then the social policy that decides autonomous action versus escalation to the human.

social-policy layer, and an injection-resistant intent catalogue. Its stance on identity and discovery follows proposals for an agent name service [5] and for authenticated delegation and visibility for AI agents [8], [9].

Messengers. Mainstream secure messengers deliver strong end-to-end encryption via the Signal protocol [12], and federated systems such as Matrix and XMPP decentralize transport [14], [15], but all move human prose between humans, with no agent mandate, no capability scoping and no structural injection defense. Table II places HAAH: it keeps the end-to-end guarantee and adds agent mediation, typed injection-resistant messages, a human-in-the-loop escalation primitive and capability-scoped authorization.

Injection and least privilege. The indirect-prompt-injection literature [6], [7] motivates HAAH's central defense; classic capability security [10], [11] shapes its authorization layer. HAAH's contribution is to combine them in a running messenger where the type boundary, not prompt discipline, is what stops injection.

VII. Discussion

Where it helps. Fronting every human with a governed agent turns three hard problems into one architecture: unwanted contact is a scope check, an autonomous assistant is a policy rule, and injection is a type boundary. The human is never spammed directly and never automated away: they are the last instance for anything irreversible, and their agent

TABLE II
QUALITATIVE POSITIONING OF HAAH AMONG MESSENGERS AND AGENT PROTOCOLS (✓ = YES, ∼ = PARTIAL, × = NO).

System	End-to-end encrypted	Agent-mediated	Typed / inj.-resistant	Human escalation	Capability-scoped	Federated / self-hosted	Primary niche
WhatsApp / Signal [12]	✓	×	×	×	×	×	human-to-human messaging
Matrix [14]	✓	×	×	×	×	✓	federated chat
Email (SMTP) [15]	∼	×	×	×	×	✓	federated store-and-forward
Raw A2A [2]	∼	✓	∼	×	∼	✓	agent task delegation
FIPA / KQML agents [3], [4]	×	✓	✓	×	×	∼	classical multi-agent systems
HAAH (this work)	✓	✓	✓	✓	✓	✓	agent-mediated human messenger

absorbs the rest. Because the human messenger layer rides the same sealed intent pipeline, the product people actually want (chat, groups, voice) inherits the same guarantees rather than bolting them on.

Characteristics of the running system. HAAH is a prototype in daily demonstrable form, not a service. It is about 8,600 lines of dependency-free Python across six modules, with 140 tests spanning RFC cryptographic vectors, protocol conformance, policy decisions, chat, groups, the Double Ratchet, DNS federation, the LLM brain, the MCP port and full end-to-end runs. Eight build-out stages (v0.3 to v0.8) each added a tested capability, from the human layer to group author signatures, the Double Ratchet, DNS federation, an LLM negotiation brain and an MCP tool-port.

Limitations and ethics. The cryptography is legible rather than hardened (not constant-time); request signing, group encryption and metadata protection are explicitly future work; there is no deployment or user study. The social-policy model gives real power to an autonomous agent, which is why every irreversible action is gated on an explicit human decision and written to a hash-chained audit the human can verify. An agent that speaks for a person also carries their accountability: HAAH's design keeps the human in authority by construction, but the governance of what an agent may promise on someone's behalf is a policy question no protocol settles alone.

VIII. CONCLUSION

HAAH treats a message not as text between humans but as an *intent between governed agents*, with the human as the last instance. Under that inversion, three long-standing difficulties collapse into one design: representing the second human keeps the task protocol honest, typed intents with a freetext-free decision path make injection a display concern, and capability scopes with a social policy make autonomy safe by default. A full human messenger, chat, groups, voice and all, rides the same sealed pipeline. The result is a zero-dependency prototype that runs, is tested against RFC vectors, and shows that the channel between personal agents can be both warm and governed. Future work hardens the cryptography, adds sender-key group encryption and metadata protection, and studies how people set policy for an agent that answers in their name.

REFERENCES

[1] Anthropic, "Introducing the Model Context Protocol," Nov. 2024. [Online]. Available: https://www.anthropic.com/news/model-context-protocol. Spec (rev. 2025-11-25): https://modelcontextprotocol.io/specification

[2] Google / The Linux Foundation, "Agent2Agent (A2A) Protocol," launched Apr. 2025, Linux Foundation project since Jun. 2025. [Online]. Available: https://a2a-protocol.org

[3] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in *Proc. 3rd Int. Conf. Information and Knowledge Management (CIKM)*, ACM, 1994. doi:10.1145/191246.191322.

[4] Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification," SC00061G, 2002. [Online]. Available: https://www.fipa.org/specs/fipa00061/

[5] K. Huang, V. S. Narajala, I. Habler, and A. Sheriff, "Agent Name Service (ANS): A Universal Directory for Secure AI Agent Discovery and Interoperability," arXiv:2505.10609, 2025.

[6] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in *Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. doi:10.1145/3605764.3623985.

[7] OWASP Gen AI Security Project, "OWASP Top 10 for LLM Applications 2025" (LLM01: Prompt Injection), 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[8] T. South, S. Marro, T. Hardjono, R. Mahari, C. D. Whitney, D. Greenwood, A. Chan, and A. Pentland, "Authenticated Delegation and Authorized AI Agents," arXiv:2501.09674, 2025.

[9] A. Chan, C. Ezell, M. Kaufmann, K. Wei *et al.*, "Visibility into AI Agents," in *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAccT)*, 2024. doi:10.1145/3630106.3658948.

[10] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[11] M. S. Miller, K.-P. Yee, and J. S. Shapiro, "Capability Myths Demolished," Johns Hopkins Univ. Systems Research Lab., Tech. Rep. SRL2003-02, 2003.

[12] T. Perrin (ed.) and M. Marlinspike, "The Double Ratchet Algorithm," Signal specification, 2016 (rev. 4, 2025). [Online]. Available: https://signal.org/docs/specifications/doubleratchet/

[13] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," in *IEEE European Symp. Security and Privacy (EuroS&P)*, 2017; ext. *J. Cryptology*, vol. 33, 2020.

[14] The Matrix.org Foundation, "Olm and Megolm," Matrix Specification. [Online]. Available: https://spec.matrix.org/latest/olm-megolm/

[15] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120, IETF, 2011.

[16] S. Josefsson and I. Liusvaara, “Edwards-Curve Digital Signature Algorithm (EdDSA),” RFC 8032, IETF, 2017.
[17] A. Langley, M. Hamburg, and S. Turner, “Elliptic Curves for Security,” RFC 7748, IETF, 2016.
[18] Y. Nir and A. Langley, “ChaCha20 and Poly1305 for IETF Protocols,” RFC 8439, IETF, 2018.
[19] H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF),” RFC 5869, IETF, 2010.
[20] A. Backman, J. Richer, and M. Sporny, “HTTP Message Signatures,” RFC 9421, IETF, 2024.