WORKING DRAFT V0.15 · NOVEMBER 2026

# Arranging People's Time: delegated scheduling authority for AI agents

Moe Loubani · Cantal · 17 October 2026

A proposed protocol, **SchCP** (the Scheduling Context Protocol), that lets an AI agent commit a person's time, with that person's signed consent, to a business or to another person, and never moves money. It is offered as a first step toward agents that coordinate people on their behalf, with an executable reference implementation, the failures it has been put through, and what remains open.

Spec: `spec/SPEC.md` (v0.4)  Reference implementation: TypeScript, ~11,200 lines

330 tests, 233 wire-level conformance vectors and a second implementation

Tested live with AI assistants on both sides

Agents can already talk, browse and pay for people. They cannot yet commit a person's time in any open, verifiable way: agree with a business, or with another person's agent, on when and where that person will be, with evidence that the person consented to exactly that. UCP's proposed Local Services council and Wix's Universal Scheduling Protocol are forming the envelope for booking a business, and AP2 and ACP carry payment. None defines what a human allowed an agent to commit to, and none covers a meeting between people.

SchCP is its own protocol for scheduling people, at a business or with each other: discovery, short holds, *mandates* (the human's signed limits, bound to the agent's key), host-signed offers accepted by their hash, and a lifecycle borrowed from iTIP. The *host* may be a business or a person whose own agent keeps their calendar, open to their contacts only, and a booking can be open-ended and happen at a place the guest names. A meeting of several people is one contract that every member accepts by its hash, by their own signature or by a rule they signed, and nothing is confirmed on anyone's calendar until all of them have. A confirmed meeting keeps its time until every member agrees to a change, and a café can be a member, its signed offer its yes. SchCP never moves money. A stated price is a signed term, paid to the host directly; most meetings have none, and money had brought most of the protocol's complexity and risk.

A TypeScript endpoint, SDK and key app run 330 tests. Six reviews and crash tests reproduced 65 defects, all fixed; many lived in the payment code v0.4 removed (§10.18). Then AI assistants used it on both sides of each booking, as guests and as hosts, for 25 rounds; what they misread or could not do changed the protocol in a few places and the assistant tools in many more (§10.21). In v0.14 the open questions the paper had carried were decided, and what the answers asked for was built: hosts state how long they keep records, receipts say how the person agreed, and SchCP stands on its own rather than as a profile of the business-booking efforts (§10.22). In v0.15 a second implementation, written in Python from the spec alone, showed where the text was not enough; once it said what was missing, a fresh one passed 218 of the 233 vectors on its first run (§10.23). §6 runs a mechanic shop's week, a sales meeting, a meeting between colleagues and a playdate for three families. **The ask** (§12): review the spec and one approval-profile decision, implement it independently from the spec alone, and pilot it with a real shop and a personal-calendar provider. §0 is a two-page brief for readers who stop there.


0 In brief
1 The problem
2 The precedent: AdCP
3 Don't reinvent the state machine
4 The landscape in October 2026
5 The protocol
6 Walkthrough and scenarios
7 Holds without squatting
8 Delegated authority
9 Distribution and what hosts get
10 Building it: tradeoffs and challenges
11 Open questions
12 Conclusion and ask
— Sources

# 0 In brief

**The problem, narrowly.** An agent can already find a slot, and Wix's Universal Scheduling Protocol (USP) and the proposed UCP Local Services council will let it hold and book one at a business. What the other side cannot yet check is *what the human authorized the agent to commit to*, and for how long: a time window, a price ceiling if there is a price, the cancellation terms the human will accept, the right to move the appointment next week. Without that, a host either takes the agent's word for it (the Personal Agent Protocol says its own approval field "works on trust") or asks the human again on its own site. Between people there is nothing at all: three parents' assistants cannot agree a Saturday playdate without the parents texting each other. The table shows what SchCP defines that those efforts do not, and the ground it shares with them.

| RULE OR FIELD | WHAT EXISTS TODAY | WHAT SCHCP ADDS | STATUS | QUESTION FOR REVIEWERS |
|---|---|---|---|---|
| **1. Scheduling constraints in a mandate:** `services`, time `window`, `max_price` | AP2 open mandates carry typed constraints, defined only for payments, and allow new types; unknown types fail.[37] USP and the charter define none for delegation. | Constraint types for an appointment, checked by the host on every action, not only at checkout. `max_price` binds to a stated all-in total; an offer with no price always fits | built | Are these the right types? An AP2 binding of them could follow (§8.6) |
| **2. No money in the protocol** | USP and the charter coordinate payment inside the booking; AP2 and ACP carry payment authority[12][15] | Nothing is charged through SchCP. A price, where stated, is a signed term paid to the host directly. Cancelling is always free and nothing is recorded; what a late cancellation or a no-show costs is the host's own business. A host that wants deposits or fees takes them in its own checkout | built | Is the separation right: scheduling commits time, payment protocols move money? |
| **3. Count:** `max_bookings` per mandate | USP: a per-buyer concurrent-hold limit (SHOULD)[36] | A per-credential count keyed by (principal, mandate id), so one consent buys one appointment; across several named hosts, a shared booking limit the key app keeps, which a cancelled booking gives back | built | Is the key app's online count worth its dependency (§11)? |
| **4. Management credential:** `bookings:[id]`, scope cancel/reschedule | USP: a booking-scoped capability credential for get, cancel and personal data on one booking, sender-constrained when issued under its DPoP-derived authentication; its security page does not say who issues it[36] | Issued by the human's key app, not the host; can only cancel or change that booking; cannot hold or book; renewed only by the key app after a verified move | built | Should the person's side, not the host, issue the credential that manages a booking? |
| **5. Approval of exact terms:** principal-signed `offer_sha256` | AP2: closed mandates signed by the user when present, or closed by the agent's endorsed key when not.[37] PAP: `approved_by` is `user` or `standing_permission`, and "doesn't prove that the User saw the terms"[39] | A stricter profile: where the host's policy asks for it, the human's key signs the hash of the host-signed offer, and the key app generates the consent text from those bytes | built | **The approval-profile decision:** when must an offer be approved by the user's key rather than closed by the agent? |

| RULE OR FIELD | WHAT EXISTS TODAY | WHAT SCHCP ADDS | STATUS | QUESTION FOR REVIEWERS |
|---|---|---|---|---|
| **6. Every agreement is a contract:** acceptances name `offer_sha256` | PAP: confirmed revisions are immutable, and only the latest can be confirmed[39] | Confirm, change, proposal and participant answers each name the hash of a self-contained, host-signed offer; changes chain by the previous hash; later policy never reaches a booking | built | Is anything missing from the contract? Each rule is a conformance vector |
| **7. People as hosts:** contacts-only calendars, open-ended meetings, named places | Calendar invitations work inside one organization; public booking pages take anyone. Neither carries a principal-signed commitment | A person's calendar as a host open to their contacts only (the list never published); bookings with a start and no agreed end; a place the guest names, signed into the offer, which the host may counter; how long the host keeps the records, stated in the offer the person approves | built | No other effort covers people as hosts: is the model right? |
| **8. Hold abuse and failure semantics** | USP: per-buyer hold limits, 5–10 minute TTLs, backoff (all SHOULD).[36] The charter names conformance vectors as a goal[12] | Controls that do not depend on identity, since identities are free: a shared cap for newcomers, standing earned by a kept appointment, attester lanes, reputation-scaled TTL. Executable cases for lost responses, replay, terms fixed across a policy change and contract mismatches | built as tests<br>wire vectors<br>untuned | Is the host-policy vocabulary right? The thresholds need real traffic |
| **9. Meetings of several people:** one contract, one hash | Calendar invitations: the organizer's copy is authoritative and each attendee answers alone.[5] USP and the charter book one business at a time | An organizer-signed contract naming every member and each calendar's signed offer. Each member accepts its hash, by their own key or by a rule they signed that anyone can check; nothing is confirmed anywhere until everyone has. Bookings say whether the business, a person or a rule said yes, and a guest can ask for the person, in a meeting member by member. A confirmed meeting keeps its time until every member agrees to a change; a café can be a member, its signed offer its yes; each member sees the others by their own contact names | built | No other effort covers meetings between people: is one contract hash the right model? |
| **Shared ground:** discovery, holds with expiry, idempotency, request signing, receipts, booking-scoped access | USP, UCP and the charter cover these for businesses | SchCP's own versions, kept close to theirs (RFC 9421 signatures, a `/.well-known` document, idempotency keys), so one endpoint could speak both | built | Is a bridge to USP or UCP worth building? |

**Two examples.** Sam asks for an oil change Thursday around 1 PM, under $100. Sam's key app signs a mandate: oil change only, Thursday 11:00–15:45, at most CAD 100 all-in, one booking, valid for an hour. The shop checks it before holding a bay and signs an offer: CAD 89.27 with tax, paid at the shop. The agent books by naming that offer's hash. The key app then issues a second credential that can cancel or move only that booking, until it ends. On Thursday morning Sam cancels: it is free, and SchCP records nothing. Whether the shop has a late-cancellation policy of its own is its business, outside SchCP (§6, §8.2).

Noor asks her assistant for a playdate with Ellis's and Priya's children on Saturday. All three parents keep their calendars on a personal-calendar service, each open only to their contacts. One mandate names the three calendars; the agent finds noon, the first time all are free, holds it at a park, and writes one contract naming the three people and each calendar's signed offer. Noor approves one sentence: a playdate with Ellis Moreau and Priya Shah, Saturday from 12:00 p.m., open-ended, at the park. Priya's assistant says yes by a rule she signed for weekend playdates, and says so; Ellis's asks Ellis, who signs a yes to the same contract hash. Only then is it confirmed on all three calendars. Nothing in it is priced (§6.4).

**Evidence, and its limits.** Built: a reference endpoint, agent SDK and key app in about 11,200 lines of TypeScript, with 330 tests; one documents a decision not to build arrival windows. The design has been through six reviews; between them they reproduced 64 defects (§10.10, §10.13, §10.14, §10.16), some of them cases the earlier fixes missed. All were fixed, each with a regression test that was checked by reverting the fix. Also built: wire-level conformance vectors with a black-box runner, an MCP server, and an optional AP2 (SD-JWT VC) mandate format (§10.17). v0.4 then took the money out, and with it the payment code many of those defects lived in (§10.18), added meetings of several people (§10.19), and left late cancellations and no-shows to the host (§10.20). Last, AI assistants (Claude Haiku) used the MCP tools on both sides, as guests and as the hosts answering them, for 25 rounds of three scenarios, with fixes after each round until three rounds in a row were clean (§10.21). v0.14 then decided the open questions the paper had carried and built the answers: a confirmed meeting changes only with every member's yes, a café can be a member, hosts state how long they keep records, and receipts say how the person agreed (§10.22). v0.15 adds an independent implementation of the agent's side and the host's checks, written from the spec alone (§10.23). Not shown: a host endpoint written by anyone else, an implementer outside the model family that wrote this one, a real business, a real personal-calendar provider, real traffic, or assistants other than one model family. Nearly every test runs this code on both sides, the conformance runner has only been pointed at this code, and the abuse thresholds are guesses.

**Who does the work, and who pays.** Three parties are needed, and none of them is a payment provider.

- An *operator* (a booking platform, an integrator, or a personal-calendar service) hosts the endpoint and its adapters and carries support.
- The host's *existing system of record*, or a person's own calendar, stays authoritative.
- An *assistant* brings demand and the consent surface.

A royalty-free protocol does not make integration or adoption free. §9.5 sets out the costs, with one hypothetical operating example.

**The ask** (§12):

1. Review the spec, starting with the rules in rows 1–9.
2. Decide row 5.
3. Implement it independently, from the spec alone, and run that implementation against the conformance vectors.
4. Pilot it with one shop and one personal-calendar provider, starting with fixed-price auto service and meetings between contacts.

A second implementation, written from the spec alone, now exists for the agent's side (§10.23); the next one should be a host endpoint written by someone else. Contact: Cantal (cantal.ca).

# 1 The problem

A personal agent can find the shop, read the reviews and tell you Thursday at 1 PM is free. It still cannot take the slot in a way the shop will honor. Every haircut, oil change and dentist visit still needs a human in the loop: calling, clicking through a booking widget, confirming a time. Between people it is worse. Two parents whose assistants both know their calendars still text each other to find a Saturday afternoon.

Phone is still the main channel. In a 2024 survey of US adults, 56% said the phone was their primary way to schedule medical appointments, and only 2% primarily used third-party booking apps.[30] What agents do today mostly follows one pattern: *discover in the agent, finish on the business's site*. Google's AI Mode, Fresha's MCP integration and the travel apps inside ChatGPT all hand the final booking step back to a web page.[17] [29] Where agents do act directly, businesses push back. Resy bans unapproved agents after one assistant pinged it hundreds of times an hour.[28]

The reason is structural. Every booking system is its own integration: Square, Mindbody, Vagaro, Booksy, Fresha, Jane, OpenTable. Many gate their APIs behind partnerships or paid tiers. Booksy's API documentation sits behind a login, and an independent catalog counts about 96 endpoints across 21 resources.[26] An agent that wants to book anywhere must integrate everywhere, so nobody does, and the human stays in the loop.

The tail is long. The US Census counts about 84,000 employer beauty salons (NAICS 812112, 2023).[33] Its Nonemployer Statistics count another 838,264 beauty salons with no paid employees (2023, all legal forms and receipt sizes).[41] The Bureau of Labor Statistics counts 175,926 private auto repair and maintenance establishments with employees (NAICS 8111, fourth quarter of 2025).[42] A non-employer business is not necessarily a booth renter, and an establishment count is not a count of owners. Still, about nine in ten salons have no staff, which suggests, but does not show, that few will ever commission an integration of their own.

No-shows are costly but hard to size. A systematic review of 105 healthcare studies averaged about 23%, across settings and years that differ widely.[34] For salons, Fresha's analysis of about three million appointments measured 3–4% of bookings, while owners in a Fresha survey reported losing about 7% of monthly revenue. Treatwell, a booking marketplace, estimates the UK salon cost at £1.6bn a year. That is a company claim whose method is not published.[35] None of these numbers measures what a protocol would recover (§9.5).

The channels that reach these businesses are priced on acquisition. Booksy's optional Boost charges 30% of an eligible new client's first visit, capped at $100, and nothing when it is switched off.[31] Fresha charges a one-time 20% fee, with a $6 minimum, on new clients it brings through its marketplace.[32] The scheduling-software market itself is small: analyst estimates of roughly $450–550M for 2025 use different definitions. [26] That is the point. The value is in who controls the booking, not in selling scheduling seats.

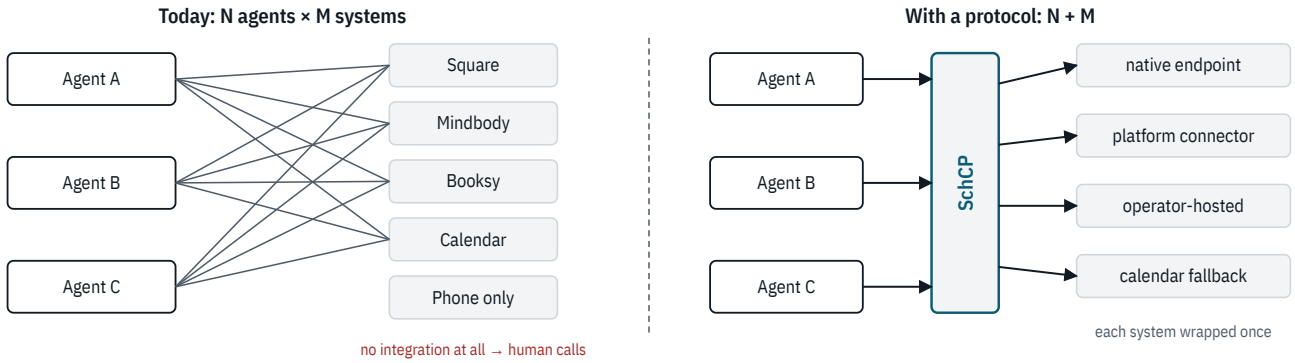



**Figure 1.** The coordination problem. Point-to-point integrations grow as agents × systems, and the long tail (calendar-only and phone-only shops) never gets one. A shared protocol grows as agents + systems, and each system is wrapped once.

This is a coordination problem, and coordination problems are solved by protocols, not features. The precise claim is narrower than "agents can't book." Agents can drive a booking widget with a browser, and voice agents can phone the shop. What is missing is a **reliable, host-sanctioned** path that has the four properties a host (a business, or a person) needs before it lets software take its time:

1. it knows *who* is asking (agent identity);
2. it knows the human actually *agreed* (delegated authority);
3. it can let an agent *hold* a slot briefly without letting it squat (bounded holds);
4. it gets an *unambiguous record* of what was agreed: the time, the place, and any price and cancellation terms.

Browser automation gives none of these. A protocol can give all four.

Two observations widen the problem beyond shops. First, much of the time agents will arrange is not for sale: a playdate, a coffee, a parent-teacher meeting, a sales demo, a viewing. Nothing about money belongs in those agreements, and a protocol that makes every meeting look like a checkout will not be used for them. Second, the other side is increasingly another person's agent, and the four questions are the same: who is asking, did the human agree, can the time be held while they decide, and what exactly was agreed. So in SchCP the party whose time is booked is a *host*, which may be a business or a person, and the protocol carries no money at all (§5.5, §10.18). It is meant as a first step toward agents that coordinate people's time, not only a business's calendar.

## 2 The precedent: AdCP

Digital advertising had the same shape of problem: buying from fifty publishers meant fifty relationships. The Ad Context Protocol launched on 15 October 2025 with six co-founders (Yahoo, PubMatic, Scope3, Swivel, Triton and Optable). It is now governed by AgenticAdvertising.org, a nonprofit trade association with four equal voting classes. The current line is 3.2.[1] It made three moves.

1. **Discovery.** A publisher hosts `/.well-known/adagents.json` listing the exact agent endpoint URLs it authorizes, scoped by property, country, time window and exclusivity. The operator's `brand.json` declares what it sells. Buyers cross-check the two files, as ads.txt and sellers.json do, and treat a missing file as unauthorized.[2]
2. **Negotiation.** Buyer and seller agents exchange terms asynchronously. Human approval is part of the state machine rather than an exception: `submitted` means the seller's own human is pending, and `input-required` means the buyer must answer. The spec says when a seller *must* go async.[3]
3. **Execution.** The agreed deal is written into the ad server through adapters, for example Prebid's open-source sales agent and its Google Ad Manager adapter.

Scheduling needs the same three moves. AdCP also gives five lessons that apply directly here.

- **Availability is not a hold.** AdCP separates a "soft avail" (a snapshot, nothing reserved) from a "firm hold" with an `expires_at`, which the seller may not resell before it lapses.[4] That is the slot-query → hold → confirm sequence this paper needs.
- **Every mutation needs an idempotency key.** A replay window must be declared, and the same key with a different body is a conflict. A retried "book" call must never become a second booking.
- **Off-protocol records must stay visible.** In AdCP's words, "creation surface is not a supported axis of authorization; account ownership is." For scheduling, phone and walk-in bookings must share the same calendar as agent bookings, or the protocol becomes a shadow ledger.
- **Describe platforms that won't cooperate without impersonating them.** AdCP's community mirrors catalogue a non-adopting platform but must list no authorized agents.
- **Stay small.** AdCP 3.2 reports 1,017 schema files, 120 error codes and a 2.8 MB schema bundle for one task in 3.1. Its own docs call a production implementation "an enterprise systems commitment."[1] A protocol meant for a two-chair barbershop cannot carry that weight.

AdCP is also a cautionary tale about legitimacy. A vendor coalition that launches outside the incumbent standards body faces a rival program (IAB Tech Lab's AAMP), questions about the founding vendor's influence, and thin early volume, whatever its technical merit.[1]

# 3 Don't reinvent the state machine

The negotiation state machine already exists and has had twenty-five years of production use. iCalendar's iTIP (RFC 5546) defines eight methods: PUBLISH, REQUEST, REPLY, ADD, CANCEL, REFRESH, COUNTER and DECLINECOUNTER. It also gives `SEQUENCE` / `DTSTAMP` ordering rules that discard stale updates, a structured status vocabulary (`4.0 Event conflict`, `3.8 No authority`), and a counter-proposal loop.[5] SchCP borrows these lifecycle concepts rather than inventing new ones, as mapped below. It does not implement iTIP as a binding. It defines its own JSON routes, ordering rules and error codes, it has no equivalent of PUBLISH, ADD or REFRESH, and it uses iTIP's status vocabulary only as a reference for its own.

| ITIP | SCHCP | NOTE |
|---|---|---|
| REQUEST, attendee replies TENTATIVE | Hold | Plus the one thing iCalendar lacks: an expiry |
| REPLY ACCEPTED | Confirm | Under a mandate; returns a signed receipt |
| REQUEST with SEQUENCE+1 | Reschedule | A stale `sequence` is rejected, as iTIP requires |
| CANCEL | Cancel | Always free, and nothing is recorded |
| COUNTER | `409 SLOT_UNAVAILABLE` with alternatives, or a host proposal | In booking, the *resource owner* is the party that counters |
| DECLINECOUNTER / REPLY | Proposal response | Accept moves the booking; decline leaves it unchanged |

iTIP lacks four things, and they are this proposal's additions.

- **Holds with expiry.** `STATUS:TENTATIVE`, `PARTSTAT=TENTATIVE` and `FBTYPE=BUSY-TENTATIVE` express uncertainty, but nothing in iCalendar expires. Every deployed system that holds slots invented its own TTL: Google's `CreateLease`, Cal.com's five-minute reservation, Wix's ten-minute `HELD`, the airline ticketing time limit.[6]
- **Verifiable delegation.** RFC 6047 says it plainly: "there is no mechanism to verify whether or not a Calendar User has authorized someone to work on their behalf. It is left to implementations."[7]
- **Discovery.** iTIP assumes you already know the other party.
- **Transport that strangers can trust.** iMIP's S/MIME signing was essentially never deployed.

> **THE ISCHEDULE WARNING**
>
> This has been tried. iSchedule (Apple and Oracle, 2009–2013) carried iTIP over HTTPS between servers, with SRV and `/.well-known/ischedule` discovery, a capabilities document and DKIM-style signing. The architecture was right. Its datatracker state is "Dead", and Apple's reference server is archived.[8]
>
> The probable causes are a bespoke crypto layer (DKIM bent onto HTTP), no incentive for the large calendar hosts, a "good enough" email fallback, and DNS operations no small business would do. There was also no commercial party whose revenue depended on it. SchCP's answers are to use standard HTTP Message Signatures (RFC 9421), let platforms answer for the hosts they serve so no salon touches DNS, and target bookings, where a business earns money when the protocol works.

For the wire format, SchCP uses JSCalendar (RFC 8984) objects for the event itself, and the protocol defines its own scheduling envelope (status, sequence, hold, mandate). The JSCalendar 2.0 draft has moved its own scheduling properties into a reserved namespace "pending further work", so depending on them now would bind the protocol to definitions the IETF itself calls unsettled.[9]

# 4 The landscape in October 2026

The first draft of this paper said the opportunity was open. A year of agent-commerce work has narrowed it, and several efforts now cover parts of the stack. The table below is the honest map.

| EFFORT | WHO / WHEN | WHAT IT STANDARDIZES | WHAT IT LEAVES TO OTHERS |
|---|---|---|---|
| **Reserve with Google** (Actions Center booking server) | Google, 2017– | Feeds plus a partner-hosted booking server: `BatchAvailabilityLookup`, optional `CreateLease` (the server may shorten the TTL), `CreateBooking`, idempotency, `PENDING_MERCHANT_CONFIRMATION` [10] | Only Google can call it, partnerships are bilateral contracts, and there is no discovery. Appointments are moving to link-out "Redirect". |
| **UCP** (Universal Commerce Protocol) | Google, Shopify and others, 2026 | `/.well-known/ucp`, RFC 9421 signatures, date-versioned capabilities, idempotency. A draft lodging-booking capability (merged 2026-09-24 as a draft contribution, not a deployed capability) has an `expires_at` soft hold[11] | The lodging draft explicitly defers post-purchase changes and booking on someone's behalf. There is no slot model. |

| EFFORT | WHO / WHEN | WHAT IT STANDARDIZES | WHAT IT LEAVES TO OTHERS |
|---|---|---|---|
| **UCP Local Services charter** | Filed by Stripe 2026-10-09; Meta, Wix, Squarespace, Shopify, Jobber, HoneyBook, Housecall Pro | Proposed council for appointments, sessions, reservations and rentals, naming USP as its starting point: "real-time slot computation … with holds". Phase 1 covers catalog, availability and holds, create, get, confirm and cancel, and payment in full, and names unknown payment outcomes and conformance vectors as concerns[12] | Still needs governance review. No-shows, reschedules and card guarantees come in later phases. No principal-issued delegation. |
| **Universal Scheduling Protocol** (USP) | Wix, Apache-2.0 draft, Aug 2026 | Service catalog, availability, holds, booking lifecycle, payment coordination; UCP-native or standalone `/.well-known/usp`.[13] Its security requirements include hold-abuse controls (per-buyer hold limits, 5–10 minute TTLs, backoff) and a booking-scoped capability credential for one booking, sender-constrained in its DPoP-derived configuration[36] | A Wix-originated draft. Its booking credential grants access to one booking but carries none of the human's limits (time window, fee caps), and the security page does not say who issues it. |
| **AP2** (Agent Payments Protocol) | Google; v0.2 April 2026, announced for donation to the FIDO Alliance[38] | Open/closed *mandates* as SD-JWT VCs bound to the agent key ( `cnf` ), typed and extensible constraints ("unknown ⇒ fail"), signed receipts. A closed mandate is signed by the user when present, or closed by the agent's endorsed key when not[37] | Only payment and checkout constraints are defined. No service time window, cancellation-fee cap or reschedule right. |
| **ACP** (Agentic Commerce Protocol) | OpenAI and Stripe, Sept 2025– | Merchant checkout sessions, plus a PSP-side `allowance` on a one-time payment token[15] | Limits live in the PSP's database. The merchant cannot verify offline who authorized what. |
| **Personal Agent Protocol** ("Poppy") | Meta and Sierra, announced 2026-10-06, Draft 0.1 | `poppy.json` discovery, agent identity, pairwise user IDs, DPoP-bound sessions, an `operations` extension for propose/confirm/cancel with immutable revisions and at-most-once execution[39] | `approved_by` is `user` or `standing_permission` , and "works on trust": a confirmation "doesn't prove that the User saw the terms or approved them." |

| EFFORT | WHO / WHEN | WHAT IT STANDARDIZES | WHAT IT LEAVES TO OTHERS |
|---|---|---|---|
| **Web Bot Auth**, Visa TAP, Mastercard Agent Pay | IETF / Cloudflare / networks | Agent identity as RFC 9421 signatures over requests, with key directories | Identity, not authority. |

Three conclusions follow. First, the envelope for booking a business is converging around USP and the Local Services council: discovery, signed requests, idempotency, holds, the booking lifecycle and booking-scoped access credentials. SchCP defines its own versions of these, deliberately close to theirs, so that one endpoint could speak both. Second, none of these efforts yet defines a host-verifiable statement of **what a human allowed an agent to commit to around a booking**. That means a mandate that can say "this oil change, Thursday 12–5, under $100, no cancellation fee", and that keeps working within those limits after the purchase, when the human wants to cancel or move the appointment. It also means an approval of exact terms whose evidence does not rest on trusting the agent. AP2 provides a credential format that could carry this, but it does not define the booking constraints. Third, all of them treat a booking as a purchase. Most meetings are not, and none of these efforts covers a meeting between two people's agents.

> **POSITIONING**
> SchCP is **its own protocol for scheduling people**: at a business, and between people's own calendars. It borrows where the efforts above fit (RFC 9421 signatures, a `/.well-known` discovery document, idempotency keys, AP2's pattern of signed limits bound to an agent key) and keeps its pieces close enough to theirs that one endpoint could speak both. What it defines that they do not is what the person allowed: limits checked on every action, offers accepted by their hash, a person's own approval of exact terms, rights to manage a booking that outlive the purchase, and meetings that are final only when every member has said yes. They cover what SchCP does not need: payment, carts and catalogs of goods. Since v0.4 SchCP carries no payment at all, and a host that wants money takes it through AP2, ACP or its own checkout. Its exact-offer approval answers a question PAP leaves to trust. The reference implementation below exists so these claims can be checked against tested behaviour rather than prose.

# 5 The protocol

SchCP standardizes payload semantics and a state machine. It does not standardize the wire, and the spec defines an HTTPS+JSON binding that could equally be carried as MCP tools or A2A messages. AdCP took the same route: it standardized the workflow, not the auction protocol underneath. Five roles take part:

- **Principal**: the human, who holds a key used only to sign mandates;

- **Consumer agent**: holds its own key and signs every request;
- **Host**: whoever's time is being booked: a business (its chairs, bays, rooms and staff) or a person whose own agent keeps their calendar. The principal on the other side is the *guest*;
- **Host endpoint**: the server that speaks SchCP for the host;
- **Operator**: the platform that runs the endpoint when the host doesn't: a booking platform for a business, a personal-calendar service for a person.

## 5.1 Discovery and endpoint authorization

A host publishes `/.well-known/scheduling.json`. It says whether the host is a business or a person, and lists services (ISO 8601 durations or `"open"`, a decimal-string price where the host states one, the resource types each service needs, where it happens), policies (hold limits, whether confirmation is instant or needs the host's approval, who may ask, and how long the host keeps booking records), the endpoint URL and the host's signing keys. A calendar that takes part in meetings of several people says so with the `meetings` capability: every person's calendar does, and a business such as a café may opt in.

Many small businesses have no website, let alone a domain they control. NFIB reports that about a third of non-employer businesses have no website; owning a domain is a stricter test, and we have no figure for it.[26] Their presence is a Booksy page, a Google Business Profile or an Instagram account. So a platform may publish an *operator document* listing the hosts it serves. The agent then applies AdCP's bilateral rule. People have no domain either; a personal-calendar service is their operator (Daybook, in the sample world).

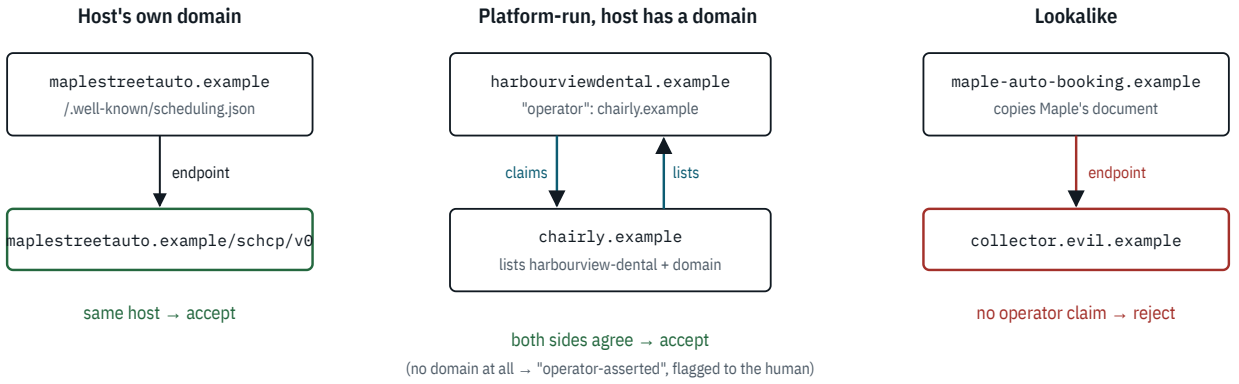



**Figure 2.** Endpoint authorization. An agent never trusts an endpoint just because a page links to it. The endpoint must sit on the host's own domain, or the host and the operator must each claim the other. A platform listing a domain that never named it is rejected too. Each case is a test in `test/discovery.test.ts`.

## 5.2 Operations

| OPERATION | NEEDS | RETURNS |
|---|---|---|
| `POST /availability` | Signed request; a contacts-only host also needs a mandate, to show who is asking | Slots with offsets, and a price and all-in total where the host states one; no `end` for an open-ended service. A snapshot, never a promise. |
| `POST /holds` | Mandate with `hold` | Hold with `expires_at` and a host-signed **offer**; or `409` plus up to five alternatives |
| `POST /holds/{id}/confirm` | Mandate with `book`, same principal; the offer's `offer_sha256` | Booking (JSCalendar event included) and a signed **receipt**; `202` if the host must approve or another participant must confirm |
| `DELETE /holds/{id}` | Owner | Released hold |
| `POST /bookings/{id}/cancel` | Mandate with `cancel` | Cancelled booking, `sequence+1`; always free, and nothing is recorded |
| `POST /bookings/{id}/change-offer` | Mandate with `reschedule`; current `sequence` | The new time, held, with a signed change offer chained to the booking's contract |
| `POST /bookings/{id}/reschedule` | Mandate with `reschedule`; the change offer's hash | Moved booking, `sequence+1`; atomic. Where the host approves each booking, the booking keeps its time and the move waits for the host's yes (§5.7) |
| `POST /bookings/{id}/proposal-response` | Mandate with `reschedule`; the proposal's hash to accept | Accept or decline a host's counter-proposal |
| `POST /bookings/{id}/respond` | The participant's own mandate; the booking's hash | A second person's answer to a booking that names them |

| OPERATION | NEEDS | RETURNS |
| --- | --- | --- |
| `POST /bookings/{id}/meeting`, `/commit`, `/meeting/withdraw` | The organizer's mandate; the meeting contract, then every member's acceptance of its hash | A meeting booking handed its contract or a revision; confirmed everywhere once every member has said yes; a change still waiting withdrawn (§5.6) |
| `POST /waitlist` | Mandate with `hold` covering the window | A waitlist entry; a freed slot arrives as a held offer |
| Webhooks to the agent | Signed by the host key | `booking.confirmed`, `booking.declined`, `booking.expired`, `booking.cancelled_by_host`, `booking.change_proposed`, `booking.change_approved`, `booking.change_declined`, `booking.ended`, `booking.no_show`, `booking.response`, `booking.accepted`, `hold.expired`, `waitlist.offered`; polling must also work |

Every request is signed by the agent's Ed25519 key using a narrow profile of RFC 9421. The signature covers `@method`, `@target-uri` and `content-digest`, and carries `created` and `nonce`. The key travels in a header, and its RFC 7638 thumbprint is the agent's identity. Every mutation carries an `idempotency_key`, which is deliberately separate from the signature nonce (§10.6).

## 5.3 State machines

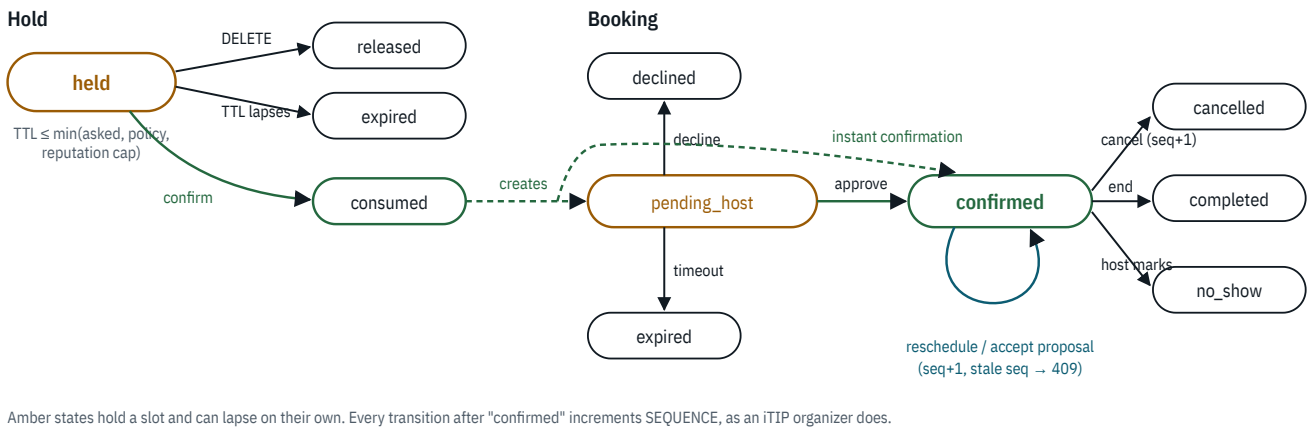



**Figure 3.** The hold and booking lifecycles. Pending-host bookings (a calendar-only business, §9, or a person who answers for themselves, §6.4) still occupy the slot until the host answers or the approval window closes. A `tentative` state has the same shape: a booking that another participant must confirm, which becomes `confirmed` when they do and `declined` or `expired` when they don't (§7.6). An open-ended booking becomes `completed` when its host ends it (§5.5).

Figure 4 draws the booking side in full, with what moves a booking and what the guest's agent hears. Two states hold time without being final: `pending_host`, where the host approves each booking, and `tentative`, where other people must still answer. Each ends `confirmed`, `declined` or `expired`, unless it is cancelled first. Live testing added one path inside `confirmed`: a guest's move waiting for the host's yes (§5.7).

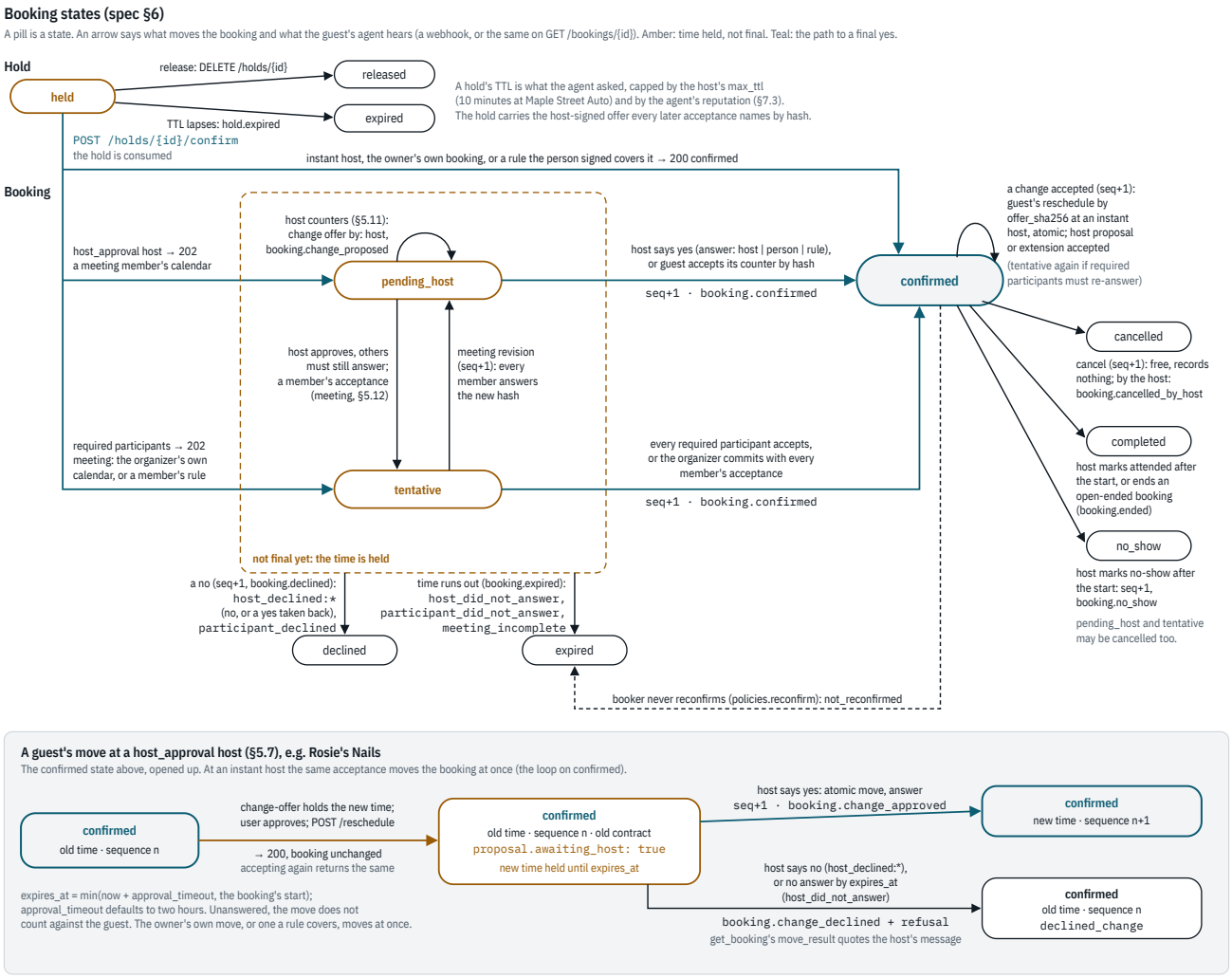



**Figure 4.** The booking states in full (spec §6). Amber states hold time but are not final; a no makes them `declined`, running out of time `expired`, and a yes or a meeting's commit `confirmed`. Every arrow names the webhook the guest's agent gets, or what it reads on `GET /bookings/{id}`. The lower panel opens up `confirmed` at a host that approves each booking: an accepted move waits as `awaiting_host` while the booking keeps its old time, and a no or no answer leaves it there, with `declined_change` saying why (§5.7).

## 5.4 Offers and receipts

Two host signatures bracket every booking. When a hold is placed, the response carries an **offer**: a JWS over the time, the place and any stated price, signed with a key from the discovery document. The agent verifies it and shows the human *those* terms, not its own paraphrase of them. When a booking is confirmed, changed or cancelled, the response carries a **receipt**, a JWS over the full booking. Since v0.14 a receipt also says how the person agreed ( `consent.approved` : they signed this exact offer, a meeting contract naming it, or only the limits their agent booked within), under which mandate ( `consent.mandate` , the one the person's key signed),

and how long the host keeps the records ( `retention` ), so that it can serve as evidence in a dispute over terms. Together they are the "unambiguous record" from §1, and the evidence for a later dispute. AP2 positions its merchant-signed checkout the same way.[14]

Since v0.3 the offer is the contract, and every agreement names one. The offer is self-contained: where the host states a price, the all-in total with its lines (service, mandatory fees and tax); the chosen provider, by the name people use ( `resource_name` , so the consent text says "Haircut with Marcus at Fades on Queen"); the place; the host's approval deadline; any reconfirmation the host will ask for; and how long the host keeps the booking's records ( `retention` ), which the consent text states ("Maple Street Auto keeps my booking details for 1 year after the booking"). SchCP sets no number for it, because laws and trades differ, and does not check that a host keeps its word. Since v0.4 it carries no cancellation or no-show terms: SchCP records neither, and a host's own policy is its business. Every acceptance carries the offer's SHA-256: a confirm, a change, a merchant proposal, a participant's answer. A different or missing hash is refused as a contract mismatch, so an agent can never commit a human to terms that differ from the bytes the host signed. A change is a new offer that names the previous one's hash, so a booking's history is a chain either side can verify. The booking keeps the terms of the offer it stands on, never the host's current policy, so a policy change or a restart does not reach it (§10.14). Terms are structured, signed data; natural-language negotiation binds nothing, and consent text is a rendering of the offer. An offer with no price has nothing about money in it at all. Offers, bookings, receipts, holds, waitlist entries and booking responses are closed in the JSON Schema: a field the spec does not define is refused, so a host cannot slip in a term the person never saw.

## 5.5 Time, place and people (v0.4)

**A start and no agreed end.** A service may have `duration: "open"` : an agreed start and no agreed end. A car dropped off for diagnosis, a playdate and a consultation that runs as long as it runs all have this shape, and v0.3 made a host invent a length for them. The host still sets time aside so that nothing is booked on top: `set_aside` as a duration, or by default the rest of the working day, up to the first close among the resources the booking takes. Slots, holds, offers and bookings carry `open_end: true` and no `end` , and a mandate's time window bounds only the start, because the start is all the human commits to. The host ends the booking when it is over, never in advance. Ending it frees the rest of the time set aside, completes the booking and sends `booking.ended` with a signed receipt that carries the real end. An open-ended booking cannot be extended: there is no end to extend.

**Where.** A service happens at the host's (the default), at the guest's address ( `location: "guest"` : a postal code and country, inside any service area the host declares), or `"elsewhere"` : a place the guest names, such as a park or a café. The `place` is part of the signed offer and becomes the calendar event's location. A meeting elsewhere with no place named is refused with the reason `place_required` .

**People as hosts.** A host has a `kind` : a business, or a person whose own agent keeps their calendar. A person's calendar can be open to their contacts only ( `policies.access: "contacts"` ). Then only the host's owner and the principals on its contact list may see its availability, hold its time, join its waitlist or confirm. Anyone

else gets `NOT_PERMITTED` with the reason `not_a_contact`, and availability needs a mandate, to show who is asking. The list itself is never published, and a contact removed between a hold and its confirmation cannot confirm.

**Money, if any.** A host may state a price. If it does, the signed offer carries it with its all-in total and lines, the consent text says it is paid to the host directly, and the mandate's `max_price` binds to the total. If it doesn't, nothing about money is in the terms, and `max_price` has nothing to bind. Fees, surcharges and per-person prices need a stated price. SchCP never carries a payment instrument: the schemas reject a request that tries.

## 5.6 Meetings of several people (v0.4)

**One contract.** A meeting is one document that every member accepts. The organizer's agent holds the same time on every member's calendar, then writes a *meeting contract* (`schcp-meeting+jwt`). It names the title, the start, the end or an open end, and the place. For each member it names their principal, their calendar and the SHA-256 of that calendar's own signed offer. A person's calendar names its owner in its offers, so the agent knows whose calendar it holds. The organizer's key app signs the contract, with a statement generated from its claims; that signature is the organizer's consent. The contract's hash is what everyone else says yes to.

**Each calendar checks its own part.** The contract travels with each confirmation. A calendar refuses it unless the guest signed it, the member named for this calendar is its owner, the offer hash is this hold's offer, and the time and place are the offer's. Every calendar checks its own entry in the same hashed document, so the contract they all accept says what every offer says, and no calendar needs another's keys.

**Who says yes.** A member's calendar waits for that member's *acceptance* (`schcp-acceptance+jwt`): a signed yes to the contract's hash. It comes from the member's own key, or from the agent that keeps their calendar under a *rule* the member signed in advance (`schcp-rule+jwt`). A rule names the agent's key, says what it covers in the person's own words, such as "Weekend playdates with my contacts", and may limit services, weekdays and organizers. A yes by rule carries the rule, so anyone can check it. Every booking says who answered: the person (`by: "person"`) or a rule (`by: "rule"`). At a business, its own answer is `by: "host"`. A guest can ask for the person, and then no rule may answer. In a meeting the organizer can ask member by member: a member's own `answers` overrides the contract's, so Ellis can be asked to answer in person while Priya's rule answers for her, and the text the organizer approves names each group ("Ellis Moreau answers in person; Priya Shah may answer by a rule they signed"). Hosts should ask their person, and use rules only where the person chose them. A rule only ever says yes; a no is always the person's own answer. The same applies to an ordinary booking at a person's calendar.

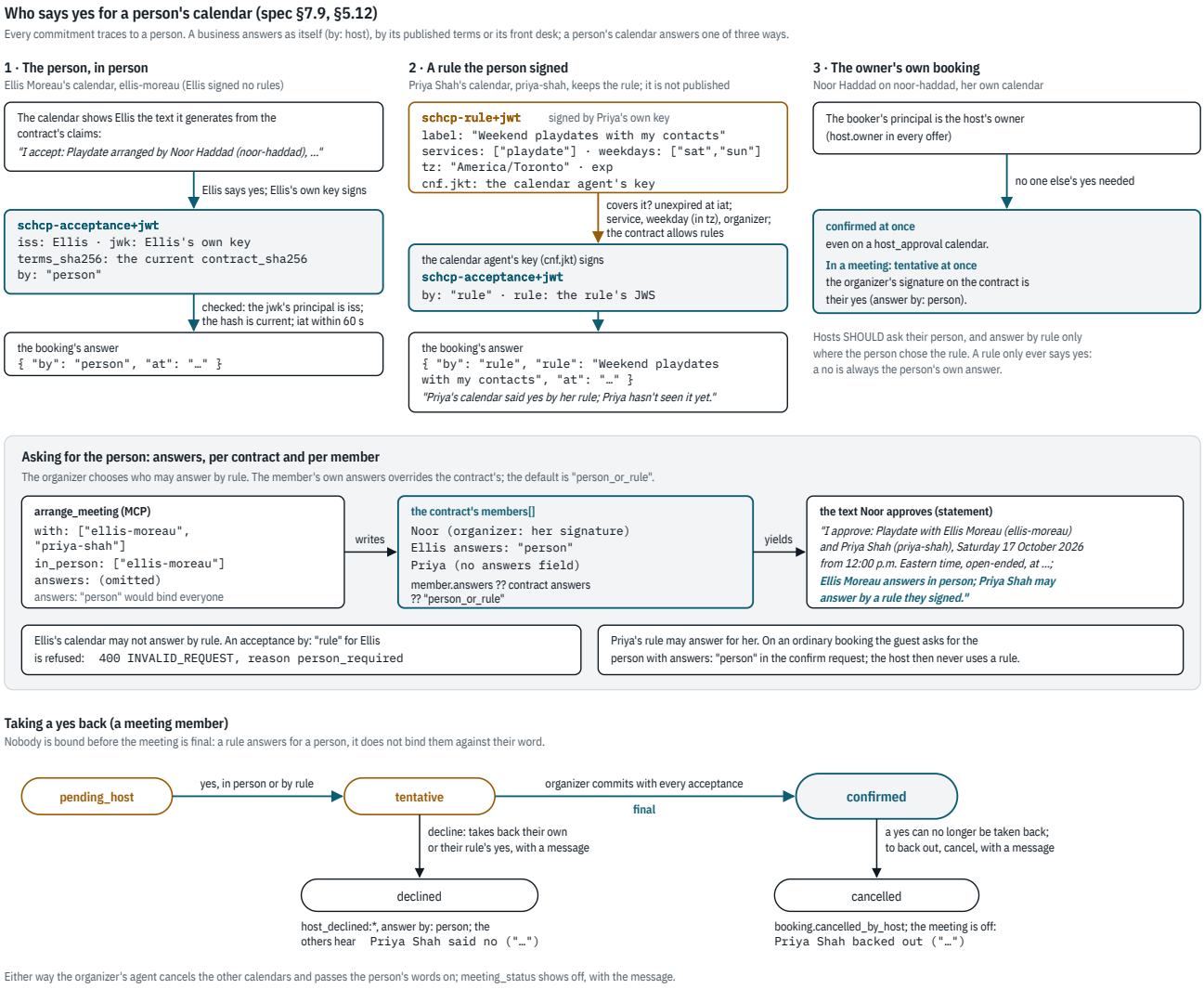



**Figure 5.** Who says yes for a person's calendar: the person, with their own key; a rule they signed, carried with the yes so another calendar can check it; or nobody, for the owner's own booking. A business's published terms are its standing yes. The lower panels show `answers` per member, and taking a yes back before the meeting is final. Both came from live testing (§10.21).

**Nothing is confirmed until everyone has said yes.** When every member has accepted, the organizer's agent brings all the acceptances to every calendar ( `commit` ). Each calendar verifies them against the same hash and confirms, and every receipt names that hash. If anyone declines, or the contract's deadline passes, everything is released, and leaving a meeting that never became final records nothing.

**A business as a member.** Three families may want the playdate at a café, with a table held for them. A business that takes part in meetings can be a member ( `business: true` , named by its host principal). It has no person to accept: its signed offer for the table is its yes, as in an ordinary booking, and a business that approves each booking says yes as the host. The meeting is final only when every person has accepted and the business has said yes, and calling the meeting off cancels the table too.

**Who the others are.** A member's calendar checks only its own entry, so the other members' names are the organizer's word. A member's agent therefore shows each other person by the name in its own person's contacts, and otherwise as someone they don't know, with the organizer's name for them only in quotes: “someone you don't know, named "Priya Shah"”.

**Counters.** A host whose booking is waiting for its answer may counter instead of saying yes or no: a different place, or a different start, as a host-signed change offer. In an ordinary booking the guest's person accepts the counter, and the booking is final, because the counter was the host's yes. In a meeting, the organizer's agent revises the contract around the counter and the organizer approves the new text once. Every member then answers the new hash; a yes to the old one no longer counts.

**Taking a yes back, backing out, calling it off.** Until the meeting is final, a member may take back a yes, their own or their rule's: a rule answers for a person, it does not bind them against their word. Their calendar declines with the person's message, and the meeting goes off. After it is final a member can still back out by cancelling their part, so a confirmed meeting can still go off. Either way the organizer's agent cancels the other calendars and quotes the member to them in their own words: "The meeting is off: Priya Shah backed out ("...")." The organizer calls a meeting off on every calendar at once, the others first, each told why in the organizer's words.

**Changing a confirmed meeting.** Until v0.14 a confirmed meeting could only be called off and arranged again, so for a moment everyone lost it. Now the organizer revises it as it revises one still gathering its yeses, and every calendar keeps the meeting confirmed at its old time and place while the revision waits beside it ( `meeting.pending_revision` ). The new time stays held until the revision's deadline, which is never after the meeting's current start. Every member answers the revision's hash. When all have, the organizer commits it everywhere and each calendar moves in one step ( `booking.change_approved` ). A no, a yes taken back, or the deadline passing drops the revision, and the meeting stays exactly as it was. This mirrors a guest's move at a host that approves each booking (§5.7). If every member said yes but one calendar can no longer move, the organizer's agent tells its person, and revises again or calls the meeting off.

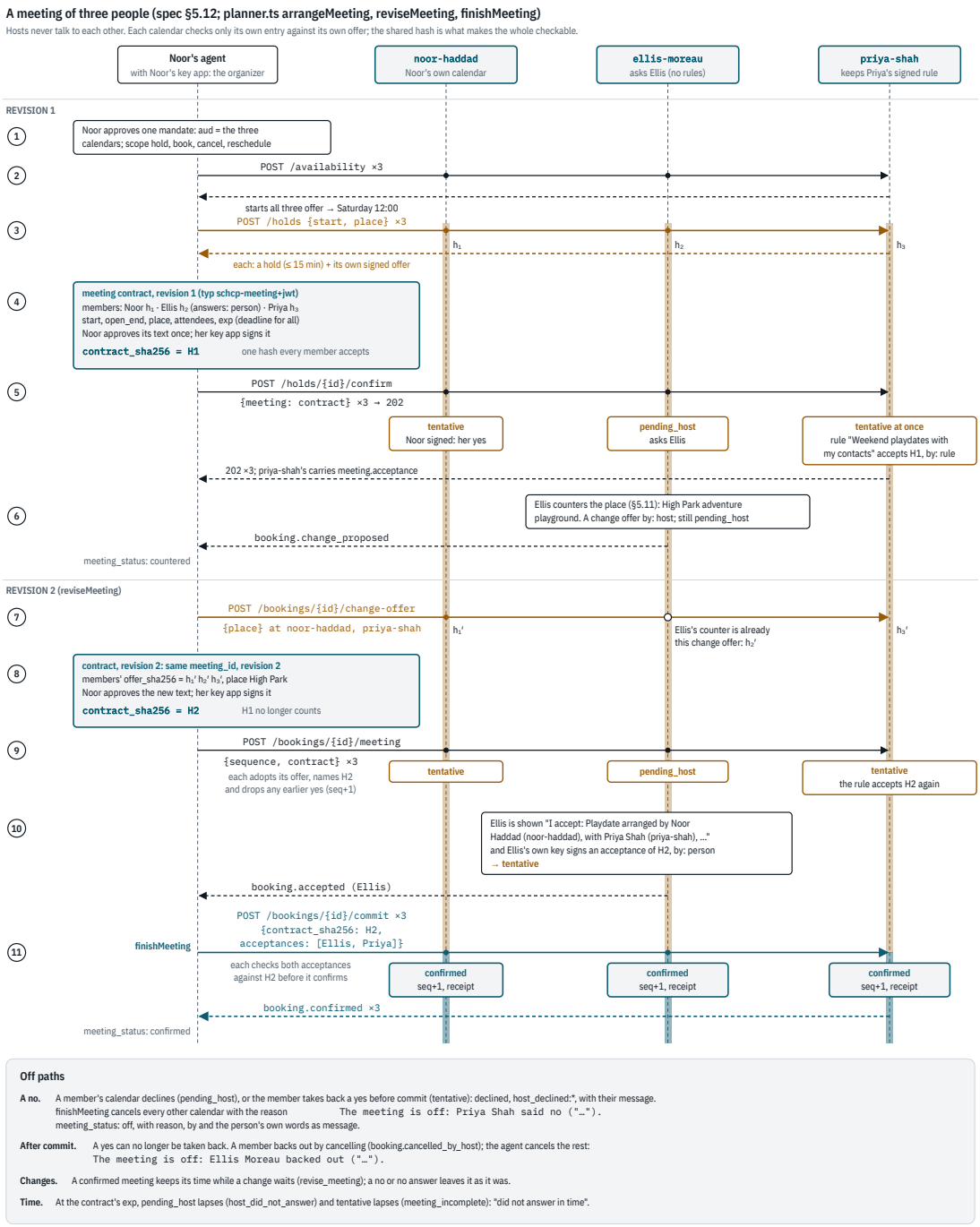



**Figure 6.** A meeting of three people, message by message: the playdate of §6.4, with Ellis countering the place. Revision 2 has a new hash, the yes to the first one stops counting, and every member answers again before the organizer's agent commits at each calendar. The inset gives the ways a meeting goes off: a no, relayed in the person's own words; a member backing out after the commit; and a change to a confirmed meeting, which keeps the old time until every member agrees (§5.6). Hosts never talk to each other.

## 5.7 Moving a booking (v0.4)

A move is not a cancellation followed by a new booking. The guest's agent asks for a change offer: the host holds the new time or place and signs an offer that names the current contract's hash. The person approves its text, and the agent accepts it by its hash. At an instant host the booking moves in one transaction (`sequence+1`, a new receipt), and a failure leaves the original intact.

A host that approves each booking approves a move too, because a move gives away a new time. Accepting then leaves the booking at its old time, contract and `sequence`, and marks the accepted offer `awaiting_host`. The new time stays held until the host's approval deadline, and never past the booking's own start. The host's yes makes the move atomically (`booking.change_approved`). Its no, or no answer, releases the new time and leaves the guest where they were, never without a booking (`booking.change_declined`, with a refusal). The booking keeps the last declined move as `declined_change`, so an agent that polls rather than taking webhooks learns why. A rule the person signed may say yes to a move it covers, the owner's own moves need nobody's yes, and a move the host never answered does not count against the guest. Live testing found this gap: the engine had moved a confirmed pedicure at Rosie's Nails, a salon that approves each booking, without the salon's yes (§10.21).

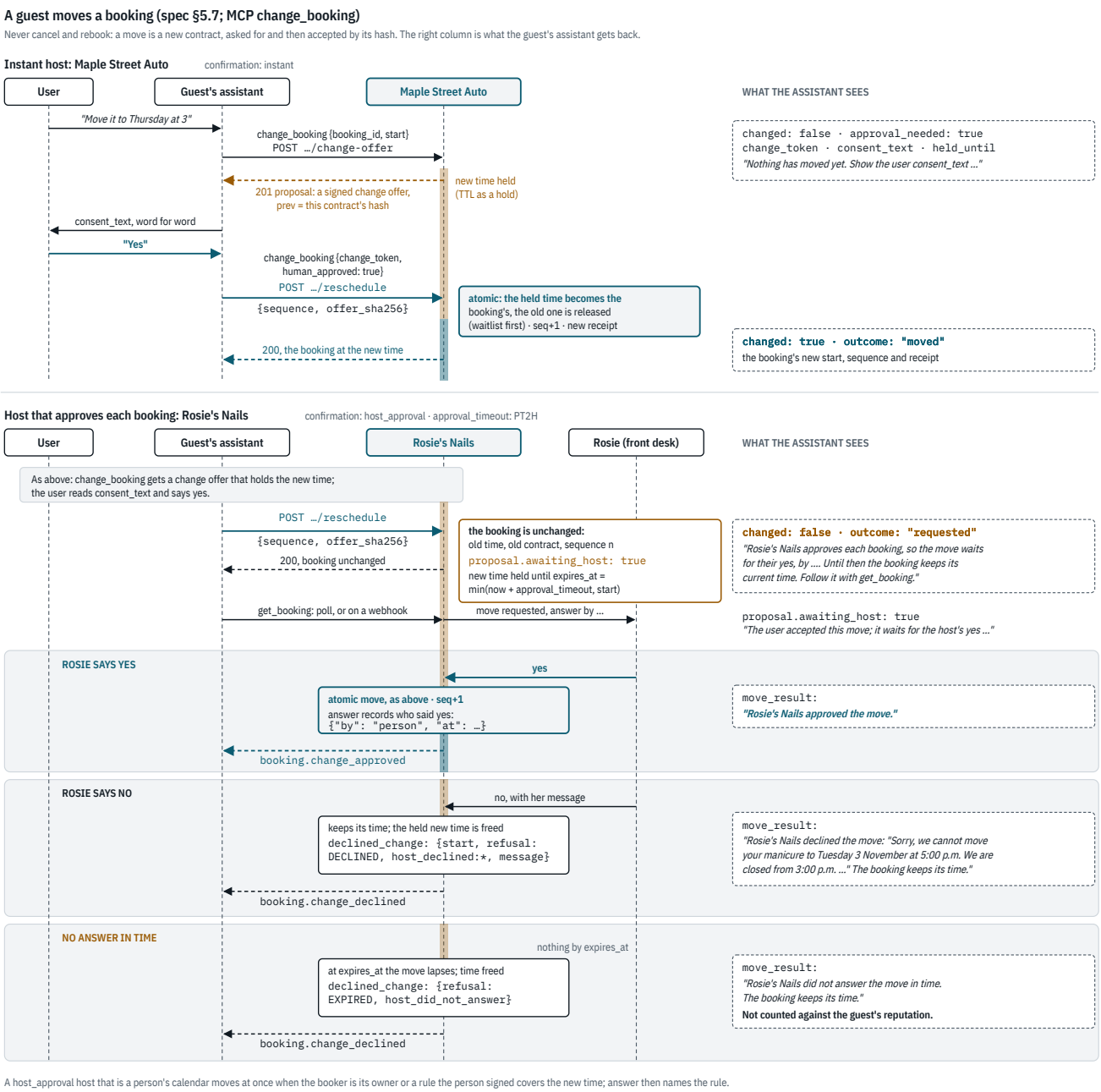



**Figure 7.** A guest moves a booking through the MCP tool `change_booking`, at Maple Street Auto, which confirms instantly, and at Rosie's Nails, which approves each booking. The right column is what the guest's assistant gets back. At Rosie's the outcome is `requested`, not `moved`, and `get_booking`'s `move_result` quotes her answer, or says she did not answer in time.

# 6 Walkthrough and scenarios: an oil change, Thursday at 1 PM

The following trace was produced by the reference implementation (`npm run demo`), not written by hand. On Tuesday 13 October 2026 at 9:00 in Toronto, Sam tells their assistant: *"Book my oil change Thursday around 1 PM, under $100."*

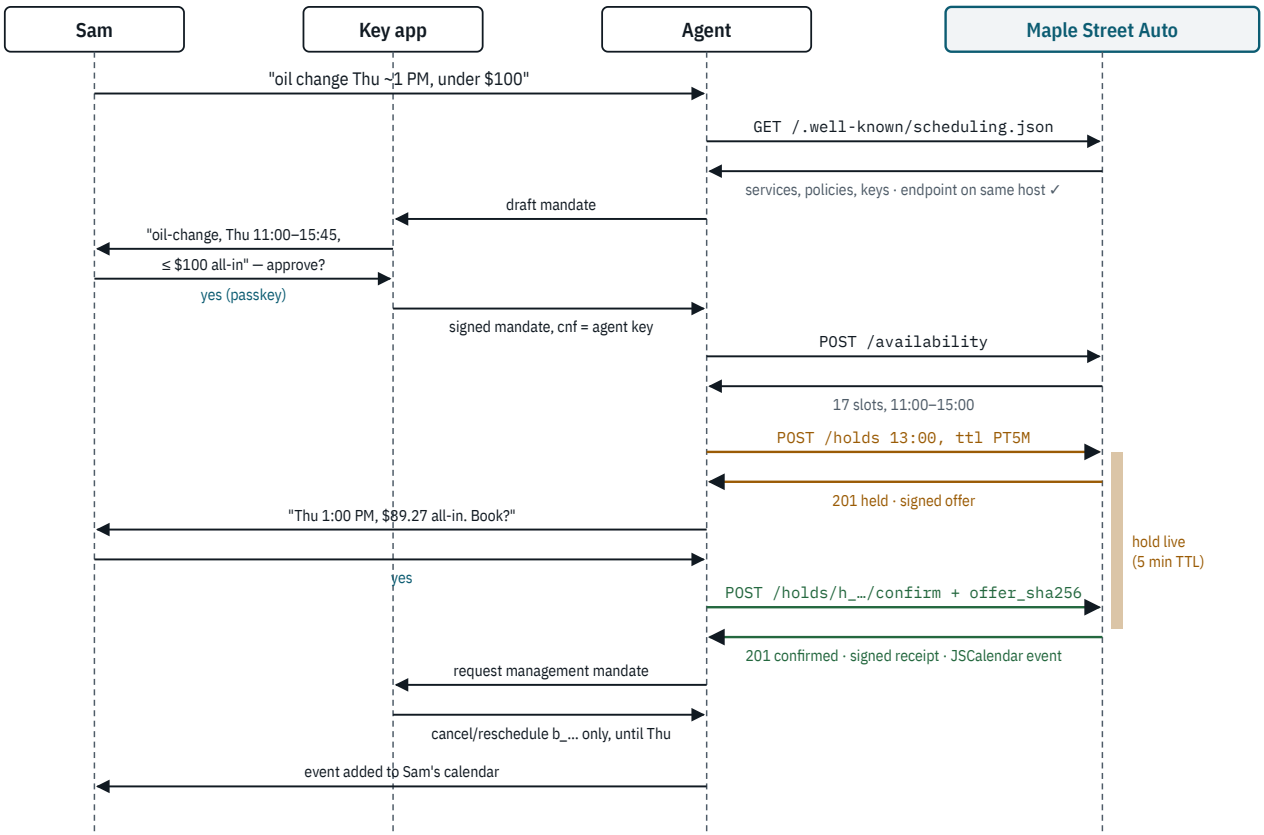



**Figure 8.** The walkthrough as a message trace. The human is consulted twice: once to approve *limits* (the mandate) and once to approve the *host's signed terms* (the offer). The hold exists so the slot survives the second conversation, and the confirm names the offer it accepts by hash.

### 1. Discovery

The agent fetches `https://maplestreetauto.example/.well-known/scheduling.json`. The endpoint is on the same host, so it is authorized. The oil change takes `PT45M` plus a `PT15M` buffer, needs one *bay* and one *technician*, and costs `79.00 CAD`. Holds are capped at ten minutes, and the shop keeps booking records for a year after each booking. Cancelling is always free, and nothing about it is recorded.

### 2. The mandate

The agent drafts limits and the key app shows them to Sam, who approves. The decoded mandate the agent receives:

```
{
  "iss": "urn:schcp:principal:jwk-thumbprint:vpxBWYd73RONoz8nFunEKSfIg6Pjlf1B_I7KDI
6hrOk",
  "aud": "maple-street-auto",
  "iat": 1791896400, "exp": 1791900000,          // one hour to buy
  "jti": "m-44291671-0285-4276-9f43-53d9e0cb6ee4",
  "cnf": { "jkt": "_ZofWgiwBCZJ6KOyQF_G7yYoHEqxNvdQRdmulbd9Yy4" },   // only this a
gent key
  "principal_jwk": { "kty": "OKP", "crv": "Ed25519", "x": "vlAzfgVf…" },
  "scope": ["hold", "book", "cancel", "reschedule"],
  "constraints": {
    "services": ["oil-change"],
    "window": { "not_before": "2026-10-15T11:00:00-04:00", "not_after": "2026-10-15
T15:45:00-04:00" },
    "max_price": { "amount": "100.00", "currency": "CAD" },
    "max_bookings": 1
  },
  "consent": { "mode": "human_present",
               "statement": "Book my oil change Thursday around 1 PM, under $100."
},
  "status": "https://keyapp-sam.example/status/m-44291671-…"
}
```

### 3. Availability

```
POST /schcp/v0/availability
Signature-Input: schcp=("@method" "@target-uri" "content-digest");created=179189640
0;
                 keyid="_ZofWgiwBCZJ6KOyQF_G7yYoHEqxNvdQRdmulbd9Yy4";nonce="a1LuJVe
EwcrLIl0cPLPvQQ";alg="ed25519"
Signature: schcp=:RPOwkcHR…:
Content-Digest: sha-256=:Ao+Cr7b5…:

{ "service_id": "oil-change",
  "window": { "start": "2026-10-15T11:00:00-04:00", "end": "2026-10-15T15:45:00-04:
00" } }

→ 200 { "timezone": "America/Toronto", "slots": [
  { "start": "2026-10-15T11:00:00-04:00", "end": "2026-10-15T11:45:00-04:00",
    "price": { "amount": "79.00", "currency": "CAD" }, "total": { "amount": "89.2
7", "currency": "CAD" } },
  … 17 slots … ], "as_of": "2026-10-13T13:00:00Z" }
```

### 4. The hold, and the host's offer

```
POST /schcp/v0/holds
SchCP-Mandate: eyJhbGciOiJFZERTQSIsInR5cCI6InNjaGNwLW1hbmRhdGUrand0In0.eyJpc3Mi…

{ "idempotency_key": "idem-ebcbe9fd-8d61-461e-b9f2-50675c9518b2",
  "service_id": "oil-change", "start": "2026-10-15T13:00:00-04:00", "ttl": "PT5M",
  "callback_url": "https://agent-sam.example/hooks/schcp" }

→ 201 { "hold_id": "h_1db43a4b-…", "status": "held",
  "start": "2026-10-15T13:00:00-04:00", "end": "2026-10-15T13:45:00-04:00",
  "expires_at": "2026-10-13T13:05:00Z",
  "price": { "amount": "79.00", "currency": "CAD" },
  "total": { "amount": "89.27", "currency": "CAD" },
  "offer": "eyJhbGciOiJFZERTQSIsInR5cCI6InNjaGNwLW9mZmVyK2p3dCIs…" }   // host-sign
ed terms
```

The offer is the contract. Decoded, it lists everything Sam would pay the shop if things go as planned (the oil change, CAD 79.00, and HST 13%, CAD 10.27, for CAD 89.27 in total, paid at the shop). Nothing in it is ever charged through SchCP (§8.2), and it states no cancellation or no-show terms: cancelling through SchCP is always free, and what Maple does about a late cancellation is its own business. Sam's limit of $100 applies to the total, not the sticker price. The key app renders it for Sam from the signed bytes:

> **WHAT SAM APPROVES**
> I approve: Synthetic oil change with Maple Street Auto, 120 Maple St, Toronto, ON, CA (maplestreetauto.example), Thursday 15 October 2026, 1:00–1:45 p.m. Eastern time, CAD 89.27 in total (Synthetic oil change CAD 79.00, HST 13% CAD 10.27), paid to Maple Street Auto directly; Maple Street Auto keeps my booking details for 1 year after the booking.

The shop's endpoint checked the mandate before it allocated anything: signature, audience, expiry, that `cnf` matches the key that signed this request, the service, the window, the price, and that this `jti` has no other live hold or booking. The allocation of a bay and a technician happened in one database transaction (§10.1).

### 5. Confirm

```
POST /schcp/v0/holds/h_1db43a4b-…/confirm
{ "idempotency_key": "idem-25a4afd7-92ae-4431-b54f-116c7b1f015c",
  "offer_sha256": "F0eLfbo_iGWnxKOSnMS943HTm-1YeMbJtDv7vYZy4No" }   // the exact of
fer accepted

→ 201 { "booking": { "booking_id": "b_1a9a1b9e-…", "status": "confirmed", "sequenc
e": 0,
    "start": "2026-10-15T13:00:00-04:00", "end": "2026-10-15T13:45:00-04:00",
    "total": { "amount": "89.27", "currency": "CAD" }, "lines": [ … ],
    "offer_sha256": "F0eLfbo_iGWnxKOSnMS943HTm-1YeMbJtDv7vYZy4No",
    "event": { "@type": "Event", "uid": "b_1a9a1b9e-…@maple-street-auto.schcp",
               "title": "Synthetic oil change — Maple Street Auto",
               "start": "2026-10-15T13:00:00", "timeZone": "America/Toronto",
               "duration": "PT45M", "sequence": 0, "status": "confirmed" } },
  "receipt": "eyJhbGciOiJFZERTQSIsInR5cCI6InNjaGNwLXJlY2VpcHQrand0Iiwia2lkIjoi…" }
```

### 6. After the purchase

The agent names the offer by its SHA-256, so the booking is exactly those bytes; a different or missing hash is refused (§5.4). Sam can cancel at any time before the start, for free, and nothing is recorded.

The buying mandate expires in an hour, but Sam may want to cancel on Wednesday. So the key app issues a **management mandate**. It is scoped to `cancel` and `reschedule` for `constraints.bookings: ["b_1a9a1b9e-…"]`, valid until the appointment ends, and it cannot place a hold or buy anything. Section 10.3 explains why this exists. In the original draft it didn't, and the tests found the gap.

## 6.1 A week at a mechanic shop

The walkthrough is one booking. A shop's week has more in it, and the test suite runs this one end to end (`test/scenarios/mechanic-week.test.ts`):

- **Tuesday.** Sam's oil change for Thursday, as above: CAD 89.27 all-in.
- **A full Saturday.** On Saturdays Maple has one bay and one technician. Kim books 10:00 for winter tires, and Priya's agent joins the waitlist for that hour. On Wednesday Kim cancels. The hour is offered to Priya as a held offer before anyone else can see it, and her agent confirms it under the mandate she already signed.
- **A parts delay.** At 8 on Thursday morning Sam's oil filter hasn't arrived. The shop proposes Friday. The proposal is a change offer chained to Thursday's contract, and Sam's agent accepts it by its hash.
- **A no-show.** The next Tuesday, Jo doesn't arrive for a three-hour detailing. The shop marks the booking a no-show; Jo's agent gets a signed `booking.no_show` and nothing else. SchCP took no deposit, states no fee and records no debt. Whether the shop bills Jo, or refuses Jo's next booking, is its own business, in its own system. A no-show does not earn Jo the standing of a returning guest (§7.2).
- **A drop-off.** "There's a noise when I brake" is a diagnosis, with no fixed price or duration. v0.4 can book it as an open-ended drop-off with no price (`test/open-end.test.ts`): the car arrives at 8:00 and a bay is set aside until the shop closes at 6. When the shop ends the booking at 10:40, the rest of the bay's day comes free, and an oil change at 1 PM that had nowhere to go now fits.
- **What it doesn't do.** The repair estimate. The quote for the work is a conversation with the shop, not a term SchCP carries, and anything that depends on parts or a mechanic's judgement hands back to staff. That is the limit of the wedge (§12).

## 6.2 A sales meeting

Nothing in SchCP requires a price. Northwind, a software vendor, publishes a 30-minute demo with a choice of account executive and no price at all. Dana, a VP, tells her assistant to book one next week with Ana, and says that Lee, her CTO, must attend (`test/scenarios/meetings.test.ts`):

- The offer states no price, so it has no total and no lines, and it names Ana: the signed contract says who the meeting is with.
- The booking names Lee as a required participant. It stays tentative until Lee's own agent, under Lee's own mandate, reads the contract and accepts it. Dana's agent cannot accept for him.
- Northwind asks bookers to reconfirm the day before. An unconfirmed demo lapses, and Ana's time comes free.
- Ana falls ill, and Northwind proposes Ben at the same time. Lee had accepted a meeting with Ana, so the change returns the booking to tentative until he accepts the new one.
- **What it doesn't do.** Qualifying questions and routing ("who covers Ontario?") stay with the vendor. The meeting link is not part of the booking. If the prospect doesn't come, Northwind can mark the booking a

no-show; what follows is up to Northwind's own system, and SchCP keeps no attendance history across hosts.

## 6.3 A meeting between colleagues

This case is the furthest from SchCP's design. Colleagues' calendars belong to people inside one organization, and calendar invitations (iTIP over email or CalDAV) already work there. SchCP adds something at the edges: when a company runs an endpoint for its meeting rooms, and when people's assistants book on their behalf. The suite runs it too:

- Mira's assistant reads three people's free/busy times from their own calendars and intersects them with the rooms' availability, on the agent's side. The first half hour all three have free is noon.
- It holds a room and confirms with Tom and Ines as required participants, on invitation terms: two hours to answer, and silence counts as yes. Tom accepts; Ines says nothing. At the deadline the booking is confirmed, and the receipt records that Ines's yes was the default.
- If Tom declines instead, the booking is declined and the room is free at once. Nobody but Tom can answer for Tom, because an answer needs his own key app's mandate.
- **What it doesn't do.** The room's endpoint cannot check that Tom is free; that is his agent's job, unless Tom's calendar is itself a host (§6.4). Among colleagues, with no room to book and no outside party, calendar invitations remain the right tool. SchCP's mandates and participant answers matter when an assistant acts for an executive, or when a meeting crosses into another organization's booking system.

## 6.4 A playdate for three families

Noor, Ellis and Priya are parents whose children want to play together. Each has an assistant that keeps their calendar on Daybook, a personal-calendar service, and each has the others on their contact list. Priya once signed a rule for her assistant: weekend playdates with her contacts may be accepted without asking her. Ellis has signed none. Nothing here is for sale. The suite runs it end to end (`test/meetings.test.ts`; the two-family version is in `test/people.test.ts`):

- **The ask.** Noor tells her assistant to set up a playdate with Ellis's and Priya's children on Saturday. Her key app signs one mandate naming all three calendars: playdates only, starting between 10:00 and 15:00 on Saturday.
- **Finding a time.** The calendars are open only to contacts, so the agent shows the mandate even to read them. Ellis's children have swimming until noon. Ellis's calendar knows that and does not say so; it simply offers nothing before 12:00. The first start all three offer is 12:00.
- **Holding it.** The agent holds 12:00 at Trinity Bellwoods Park playground on all three calendars. A playdate has no agreed end. Each calendar sets aside three hours and signs an offer with a start and a place, and no end and no price.

- **One contract, one sentence.** The agent writes one contract naming the three people, their calendars and the hash of each calendar's offer. Noor's key app renders it: “I approve: Playdate with Ellis Moreau (ellis-moreau) and Priya Shah (priya-shah), Saturday 17 October 2026 from 12:00 p.m. Eastern time, open-ended, at Trinity Bellwoods Park playground, Toronto.” Her signature on the contract is her yes.
- **Three calendars, three answers.** Each calendar checks its own part of the contract against its own offer. Noor's own calendar has her signature. Priya's rule covers a weekend playdate, so her assistant says yes for her, and the booking records that the rule "Weekend playdates with my contacts" answered, not Priya. Her assistant can tell her she hasn't seen it yet. Ellis's assistant asks Ellis, in words written for Ellis: “I accept: Playdate arranged by Noor Haddad (noor-haddad), with Priya Shah (priya-shah), Saturday 17 October 2026 from 12:00 p.m. Eastern time, open-ended, at Trinity Bellwoods Park playground, Toronto.” Ellis signs a yes to the contract's hash.
- **Only then, everywhere.** Until Ellis answers, nothing is confirmed on any calendar, and a commit that tries is refused with the names of those who have not answered. Then Noor's agent brings Ellis's and Priya's signed yeses to all three calendars. Each checks both against the same hash and confirms, and the three signed receipts name the one contract.
- **A counter.** In another run, Ellis would rather meet at Christie Pits. Instead of saying yes or no, Ellis's assistant counters with a signed offer for the new place. Noor's agent asks the other two calendars for the same change and writes revision 2, and Noor approves the new sentence once. The hash has changed, so everyone answers again: Priya's rule at once, Ellis by signing again. A yes to the first contract no longer counts.
- **A no, or silence.** If Ellis declines, Noor's agent cancels the other two bookings. They never became final, and everyone's Saturday is free again. If not everyone has answered by the contract's deadline, the meeting lapses on every calendar.
- **Ending it.** At 2:10 p.m. the children are done. Ellis's assistant ends the booking on Ellis's calendar; the receipt records the real end, and the rest of the afternoon comes free.
- **A stranger.** An agent for someone who is not on Ellis's list sees no availability and can hold nothing: `NOT_PERMITTED`, `not_a_contact`. The list itself is never published.
- **A change after it is confirmed.** On Friday Priya asks to start an hour later. Noor's agent writes a new revision and Noor approves it. The playdate stays confirmed at its time on all three calendars while Ellis and Priya answer. If either says no, or not everyone has answered before the playdate starts, it stays as it was. Only when both have said yes does it move everywhere at once (`test/meeting-changes.test.ts`).
- **At a café.** In another run the friends meet for coffee at Bellwoods Café, which holds a table as one more member. The café's signed offer is its yes, and nothing is final until the people have said yes too (`test/scenarios/cafe-meetings.test.ts`).
- **What it doesn't do.** The agent did not choose the park: Noor named it, and Ellis countered. Where to meet is the people's own business, like getting there. It will not arrange the next playdate in two weeks: recurring meetings are not in v0.4, so each one is arranged on its own.

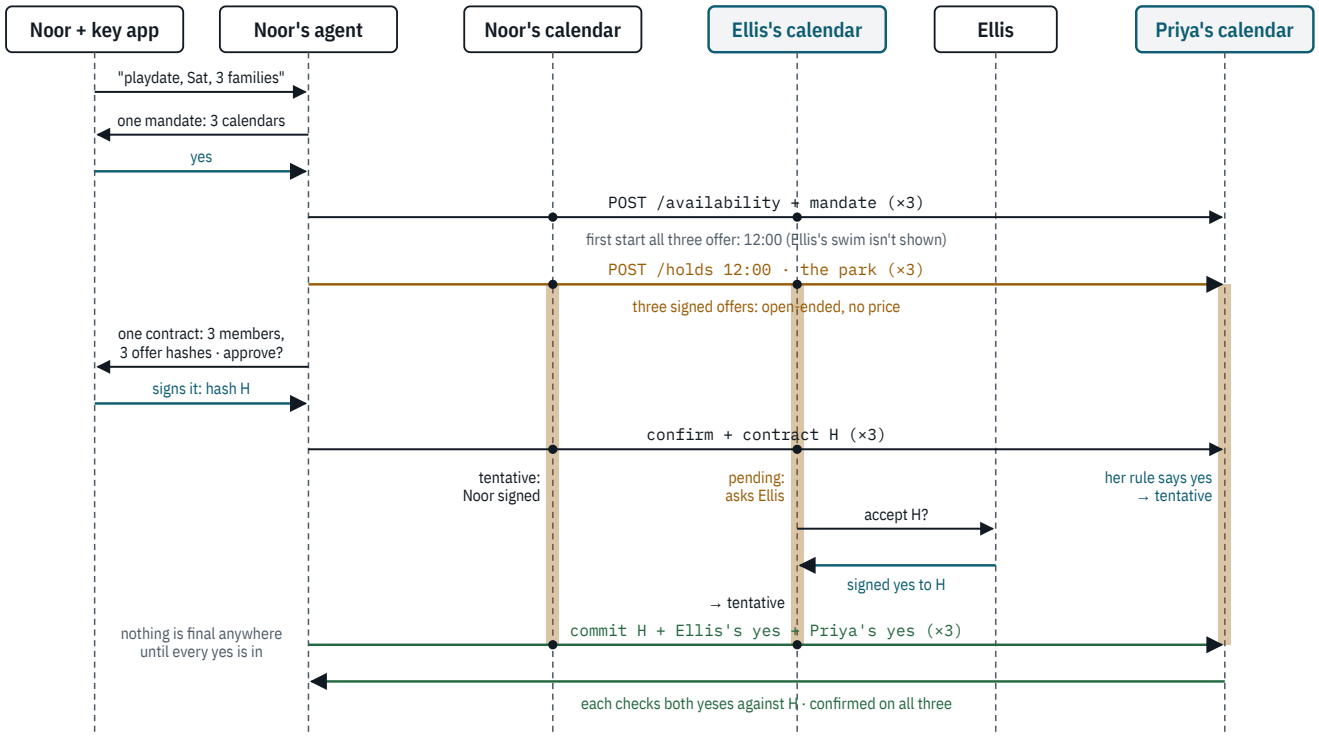



**Figure 9.** A playdate for three families. One mandate names three calendars. The agent holds the same start on each and writes one contract naming every member and each calendar's signed offer, and Noor's signature on it is her yes. Priya's calendar answers by a rule she signed; Ellis's asks Ellis, who signs a yes to the same hash. Only then does Noor's agent commit, and every calendar checks every yes against that hash before it confirms. Each step is in `test/meetings.test.ts`.

## 6.5 Where it fits

| SCENARIO | FIT | WHAT CARRIES IT | WHAT'S MISSING |
|---|---|---|---|
| Haircut, nails, oil change, tire swap | fits | Holds, signed offers, attendance the host declares, waitlist | — |
| A child's dental visit | fits | A named attendee who isn't the signer; the parent's closed approval; a refusal of free-text notes that points to the clinic's intake form; the clinic's records retention, stated in the offer the parent approves | — |
| Sales demo | fits | Unpriced offers, required participants, reconfirmation | Qualification and routing stay with the vendor |

| SCENARIO | FIT | WHAT CARRIES IT | WHAT'S MISSING |
|---|---|---|---|
| A playdate or a coffee between friends | fits | People as hosts with contacts-only calendars; a named place the other side may counter; an open end and no price; one contract every member accepts by its hash; a change after it is confirmed keeps the old time until everyone agrees | Recurring meetings, not in v0.4 |
| A coffee at a café, with a table held | fits | The café as a meeting member, its signed offer its yes; final only when every person has said yes | — |
| Scarce slots: a restaurant's 10:00 drop, a popular barber | fits | Release schedules, party size with combinable tables, waitlist, newcomer pools, reputation, a walk-in reserve | Thresholds tuned on a real drop (§11) |
| A deposit to secure a scarce slot | outside SchCP | The host's own checkout and its own terms; SchCP carries only a no-show the host declares | By design (§7.4) |
| Meeting between colleagues | partly | Room holds, participant answers, delegation | Inside one organization, invitations already work |
| Fitness class with 20 seats | fits | Shared capacity: a room and an instructor each take 20 | Passes and memberships stay with the gym, by design |
| A plumber at a set time, at home | fits | A service address checked against the service area; a same-day surcharge in the total | Getting between jobs stays the plumber's own planning, by design |
| A plumber with a two-hour arrival window | no | — | Bookings are for exact times, by design |
| A drop-off for diagnosis; other open-ended jobs | fits | An open-ended booking with no price; a bay set aside until close; the host ends it and the rest of the day comes free | The repair estimate stays a conversation |

# 7 Holds without squatting

A hold is the key primitive and the main abuse surface. iTIP named "flooding a calendar" as a threat in 2009.[5] In booking, the same attack is slot denial: an agent places holds it never intends to confirm, at scale. Airlines have fought a version of this for decades. "Churning" is repeated cancel-and-rebook to extend a ticketing time limit, and carriers answer it with debit memos charged to the travel agency.[6] The threats SchCP has to stop:

- **Fan-out.** One agent holds every Saturday slot "just in case."
- **Sybil agents.** Agent keys are free to mint, so per-agent limits alone mean nothing.
- **Hold-and-walk-away.** Holds that are never confirmed and never released.
- **Scraping.** Availability polled continuously.
- **Resale.** Scarce slots held and sold on, the "digital velvet rope" problem restaurants cite for gating their APIs.

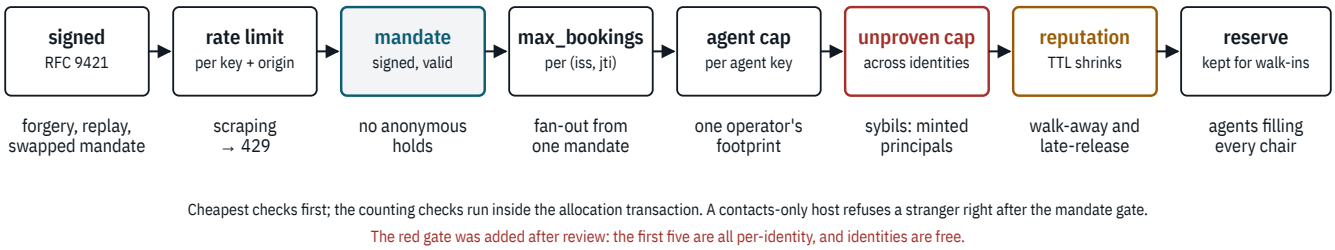



**Figure 10.** The hold gauntlet in the reference endpoint. The mandate gate ties every hold to a principal and a countable `jti`. But a self-certifying principal key costs nothing to mint, so per-identity limits alone do not stop a script from holding a whole day. The cap on concurrent holds from principals that have not yet kept an appointment here bounds that attack (§7.2). Since v0.2, newcomers vouched for by an attester the host trusts draw on a separate pool of the same kind (§7.5). Until v0.4 the last gate was a deposit; it is gone with the rest of the money (§7.4).

## 7.1 Proof of intent beats proof of work

Proof-of-work puzzles are cheap for a well-funded operator and expensive for a phone. They are a speed bump at most. Proof of intent is stronger: SchCP requires a valid mandate before any hold, and each hold counts against that mandate's `max_bookings`. A mandate for one oil change can hold one slot. The fan-out test in the suite gets `MANDATE_LIMIT` on its second hold.

## 7.2 The sybil problem, and a bound for it

The first draft of this section said the mandate gate turns "a thousand holds" into "a human signing a thousand mandates." The independent review (§10.10) showed that claim was false. In v0.1 a principal is identified by its own key, and nothing stops a script from minting a fresh principal and a fresh agent key for every hold. That bypasses every per-identity control: the per-agent cap, `max_bookings`, per-key rate limits and reputation. In the reviewer's proof of concept, 36 throwaway identities held every slot of the barbershop's Thursday in 349 ms, and re-holding every five minutes kept it blocked.

The structural fix is to make a principal cost something: an attestation from a party the host trusts (§7.5), or, outside SchCP since v0.4, a deposit a business takes from first-time customers in its own checkout. Underneath both, the reference endpoint keeps one control that does not depend on identity at all. `max_active_unproven` caps how many holds can be live *at once, across all identities*, for principals that have not yet kept an appointment with this host. With a cap of six at Fades on Queen, the same attack occupies six slots out of 36, and a returning customer is never affected. Standing is earned only by a kept appointment: a booking whose end has passed without being cancelled or marked a no-show. In v0.5 any confirmed booking counted. The deep review showed that a script could book and cancel for free to promote its throwaway identities (§10.14). Releasing a hold in the last fifth of its TTL now counts as an expiry, so "release just before it lapses" no longer launders reputation. The cost is that during an attack, genuine first-time customers compete for those six concurrent holds. That is a real tradeoff, and it is why the cap is a merchant policy rather than a protocol constant. Since v0.2, newcomers who carry a trusted attestation are in a separate pool and are not affected (§7.5).

## 7.3 Reputation-scaled TTL

The endpoint tracks, per agent key *and* per principal, the share of holds that expired without being confirmed or released. The TTL it grants shrinks as that share grows, and eventually holds are refused. The curve in the reference implementation:

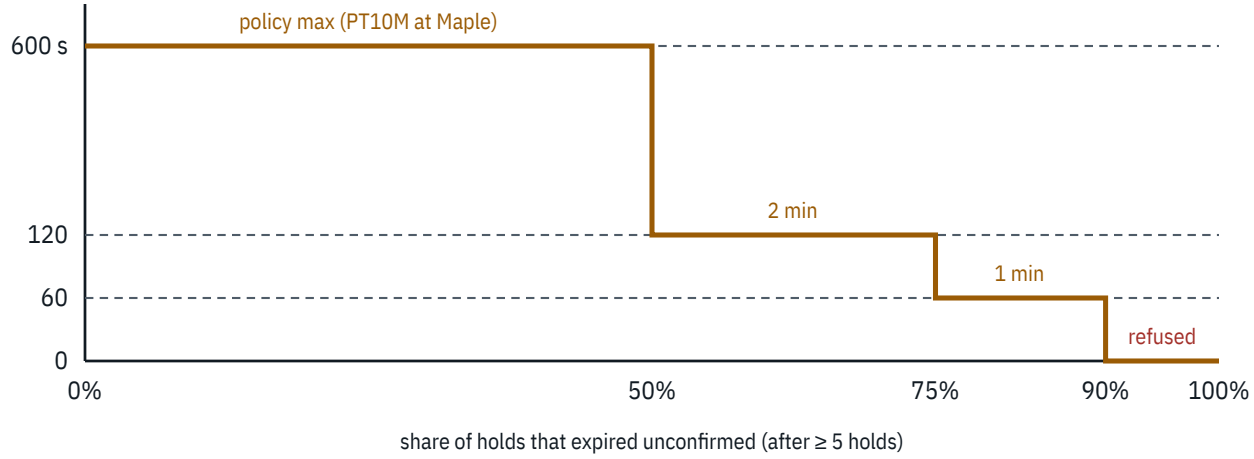



**Figure 11.** Maximum hold TTL by expiry rate. Measured on the reference endpoint: an agent that abandons every hold gets five ten-minute holds and `HOLD_LIMIT` from the sixth on, because its expiry rate jumps straight to 100%. The intermediate tiers are for agents that convert some holds and abandon others; one that abandons three in five gets two-minute holds. A well-behaved agent asking for the same slot is unaffected. Rotating to a fresh agent key does not help, because the principal is tracked too.

v0.1's reputation had no time decay, so a suspension never lifted. Adding decay was subtler than applying a half-life to the counts, for a reason §10.12 explains. The reference endpoint now keeps exponentially decayed counts, with a half-life of seven days, and never divides by fewer than `min_sample` outcomes. With the defaults, an agent suspended after five straight expiries gets 60-second holds back after about a day, 120-second holds after three days, and full-length holds after a week. Whether a principal has *ever* kept an appointment here is stored separately and never decays, so a regular from last year is still a regular.

Two limits remain. Reputation is per host, which is easy to keep but slow to learn. And it measures outcomes, not intent: it cannot tell a careless agent from a hostile one.

## 7.4 Scarce slots without deposits

Up to v0.3 a service could require a deposit: a payment authorization on the hold, captured at confirm, refunded on cancellation. For a three-hour detailing slot it was the strongest protection against a hold that turns into a no-show. v0.4 removes it with the rest of the money (§10.18). What protects scarce slots now is everything that moves no money:

- short holds whose TTL shrinks with the agent's and the principal's record of abandoning them (§7.3);
- a shared cap for newcomers, with a separate lane for those an attester vouches for (§7.2, §7.5);
- a walk-in reserve that keeps some resources out of agents' reach, and release schedules that open a day at once (§10.17);
- a waitlist that offers a freed slot to the next person, as a hold, and reconfirmation the day before, where silence can release the slot (§7.6);

- a no-show the host declares, which earns the guest no standing as a returning guest (§7.2). Anything else that follows, a fee or a refusal next time, is the host's own business, in its own system.

A business that still wants a deposit takes it in its own checkout, outside SchCP, and the protocol does not vouch for it. That is a real loss for the scarcest slots. The payments-risk seat's verdict on v0.3 was to underwrite deposits at confirm (§10.13); v0.4's answer is that a scheduling protocol is the wrong place to underwrite anything. Card networks show what the other answer costs: a card-not-present authorization lasts about seven days, so a deposit for an appointment further out needs a stored credential and the customer's agreement to merchant-initiated charges.[14] Whether these controls hold no-show rates where deposits did is a question for a pilot (§11).

## 7.5 Attestations: making a principal cost something

§7.2 ended on the structural problem. A principal key is free, so the unproven cap must treat every newcomer as a possible script, and during an attack real first-time customers queue behind the flood. v0.2 lets a host tell some of them apart without appointing a gatekeeper. An *attester* checks something scarce, such as a phone number, a payment card or an ID document, and signs a short statement binding that check to one principal key: `{iss: "phonecheck.example", sub: <principal>, assurance: "unique_phone", exp: …}`. The key app attaches it to the root mandate.

Four rules keep an attester from becoming a toll booth:

- **The host picks its attesters.** They are listed in its own discovery document under `trust.attesters`. The protocol names none.
- **Attestations only add.** An unknown attester, a bad signature, an expired statement, or one about a different key is ignored, not rejected. A guest without an attestation is exactly where v0.1 left them.
- **They buy one thing.** An attested newcomer's holds count against that attester's own pool (`max_active_per_attester`) instead of the shared unproven pool. Every other control still applies.
- **Attesters have reputation too.** If an attester's vouchees keep abandoning holds, its TTL cap reaches zero and its attestations stop counting. Its users fall back into the ordinary pool rather than being locked out.

In the test world, every sample host trusts a simulated phone-verification service that vouches for one key per number. With the new-customer pool at Fades on Queen full of throwaway identities, an unverified newcomer is refused and a verified one gets a hold. Copied attestations, an impostor signing under PhoneCheck's name, and expired statements all add nothing. An attester whose users squat loses its standing after five abandoned holds.

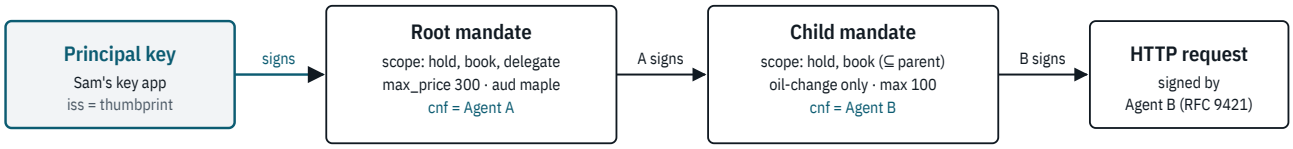
The costs are real. A phone number is scarce, not expensive: SIM farms exist, so an attester is only as good as its checks. That is why its standing is earned with each host rather than granted. An attestation is also a stable identifier. Two hosts that see the same principal key and the same attester can link a person's visits. A privacy-preserving version needs per-host keys and attestations, for example an SD-JWT or BBS+ profile, and v0.2 does not attempt it.

### 7.6 Everything that locks a time slot is a hold

v0.3 adds two new ways to occupy a slot, and each could have become a free lock. A **waitlist** entry needs a mandate covering its whole window and counts against that mandate's `max_bookings` and a per-agent cap. When a booking ends early, the endpoint offers the time, in the same transaction and on the same chair where it can, to the first waiting entry it suits, as a hold the host places; an offer that lapses counts against the waiter and moves down the line. A **tentative** booking, one that a second person must confirm from their own key app, occupies its slot until they answer, counts against the booker's hold caps meanwhile, and a lapse counts like an expired hold. The same mechanism lets a host ask the booker to reconfirm the day before.

## 8 Delegated authority

"Book on my behalf, up to $50" needs a trust framework, not just a token format. The industry has converged on one pattern, which AP2 v0.2 and Mastercard's Verifiable Intent both follow. The user approves constraints once on a non-agentic surface. The result is a signed credential bound to the agent's key. The verifier evaluates the constraints in deterministic code on every action and returns a signed receipt.[14] SchCP's mandate is a booking-shaped instance of that pattern.




**The host checks, on every action**

1 root signature verifies; iss = thumbprint(principal_jwk)
2 each child signed by the key its parent's cnf names
3 child scope ⊆ parent scope, child exp ≤ parent exp, depth ≤ 2
4 aud names this host (no wildcards unless opted in)
5 leaf cnf = the key that signed this HTTP request
6 the action satisfies the constraints of every link
7 live holds + bookings under each jti < max_bookings
8 not revoked (status URL, cached 30 s)

Rule 6 avoids a "constraint subset" algebra: a child can claim max_price 5,000, but the parent's 300 still applies.

**Figure 12.** Mandates, proof of possession and attenuation. A stolen mandate is useless without the agent key it is bound to (check 5). A sub-agent can be given less authority, never more (checks 3 and 6). All eight checks are exercised by `test/mandates.test.ts`.

### 8.1 What a mandate says

The constraint vocabulary is deliberately small: `services`, a time `window`, `max_price` (which binds to a stated all-in total; an offer with no price always fits), `max_bookings`, `bookings` for management mandates, and `offer_sha256` for closed mandates (§8.5). One more came from the reviews: `shared_limit: true`,

which makes `max_bookings` one limit the key app keeps when `aud` lists several hosts, and which a cancelled booking gives back. v0.4 removed `max_deposit`, the long-deprecated `max_cancellation_fee` and `max_no_show_fee`, a ceiling the human once approved in advance for a stated no-show fee (§10.17): there is nothing left for them to cap. For an open-ended booking the window bounds the start only. Times are RFC 3339 with offsets. "Thursday afternoon" is resolved to instants on the consent surface *before* signing, because that is where the human confirms what the words mean. Money is a decimal string plus an ISO 4217 code, and a mandate in USD never authorizes a price in CAD: there is no implicit conversion, and a test checks it.

## 8.2 Human-in-the-loop escalation

Mandates are bounded on purpose, so agents will run into their limits, and the protocol makes that a typed conversation rather than a failure. Up to v0.2 the hard case was a late cancellation: the host quoted a fee, the agent's mandate allowed none, and the agent had to go back to the human before it could even cancel. v0.3 moved the money question to the human's next offer: a recorded amount appeared there as an adjustment that only the human's own signature could accept. v0.4 takes the question out of SchCP altogether. Cancelling is always free, never waits for approval, and records nothing. What a late cancellation means to the host is between the host and the guest, outside the protocol. What remains of the escalation is the closed approval (§8.5): a host can require the human's own signature on every offer, and the agent asks at the confirmation step rather than after a refusal.

The other side has a person in the loop too. When a host is a person, its agent asks that person before saying yes, unless a rule the person signed covers the request, and the booking records which it was (§5.6). A guest who wants the person rather than a rule can ask for that, and then the host's agent may only ask. A rule is the person's signature made in advance. The difference between "the agent decided" and "the person decided earlier" is whether anyone else can check it.

**What an assistant shows.** With no price to anchor attention, a person may approve everything unread, and the consent sentence is the protocol's whole user interface. Since v0.14 the spec asks assistants to show the exact consent text, to point out what differs from what the person asked for (another time, place or person, a price they did not mention or above their limit) and how long the host keeps the records, and never to say yes for the person except under a rule or limits they signed. The MCP server's `hold_slot` returns those differences as `differs_from_request`.

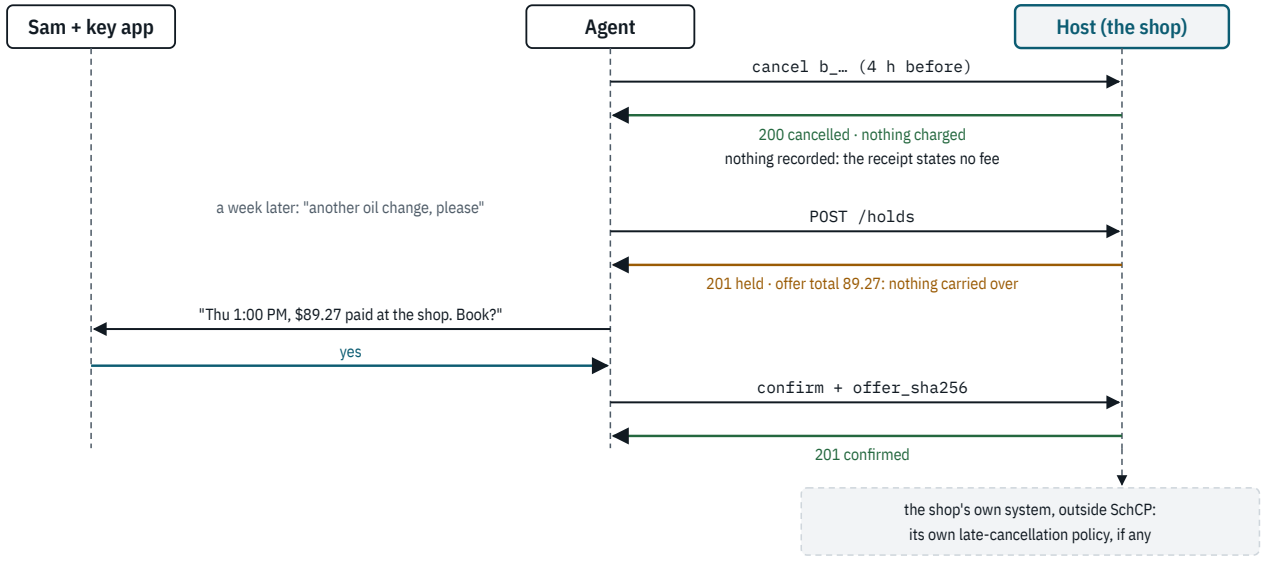



**Figure 13.** In v0.4 neither the cancellation nor the next booking is a money moment. The late cancellation is free, and SchCP records nothing about it; the next offer is priced as if it had never happened. Any policy the shop has for late cancellations is its own, kept and applied outside SchCP. In v0.3 the amount was recorded and came back on the next offer as an adjustment that only Sam's signature could accept; §10.18 and §10.20 explain why each went. The amounts are the reference implementation's.

## 8.3 Management mandates

A mandate short enough to be safe for *buying* has usually expired by the time the human wants to *change* what they bought. The answer is a second, long-lived but tightly scoped credential, issued at confirmation time. Its scope is `cancel` and `reschedule`, it carries `bookings: [id]`, and it lasts until the appointment ends. It cannot hold, buy, or touch any other booking. This was not in the original design. Section 10.3 describes how it was found.

## 8.4 Revocation

Short expiry is the primary control. Mandates may also carry a `status` URL that the host checks before mutating actions, with a 30-second cache so a slow key app cannot take bookings down. Revoking a mandate stops *future* actions. It does not cancel a confirmed booking, which is an explicit action with its own terms. PAP's rule for operations is the same.[16]

## 8.5 Closed mandates: the human signs the offer

An open mandate says what the agent *may* agree to. It cannot show what the human *did* agree to. The host can check that the price and slot fit the limits, but not that the human ever saw these exact terms. For a clinic, or any host whose own policies will bind the guest outside the protocol, that gap is where disputes happen. AP2's answer is the closed mandate, a credential over the specific cart.[14] SchCP v0.2 builds the booking version.

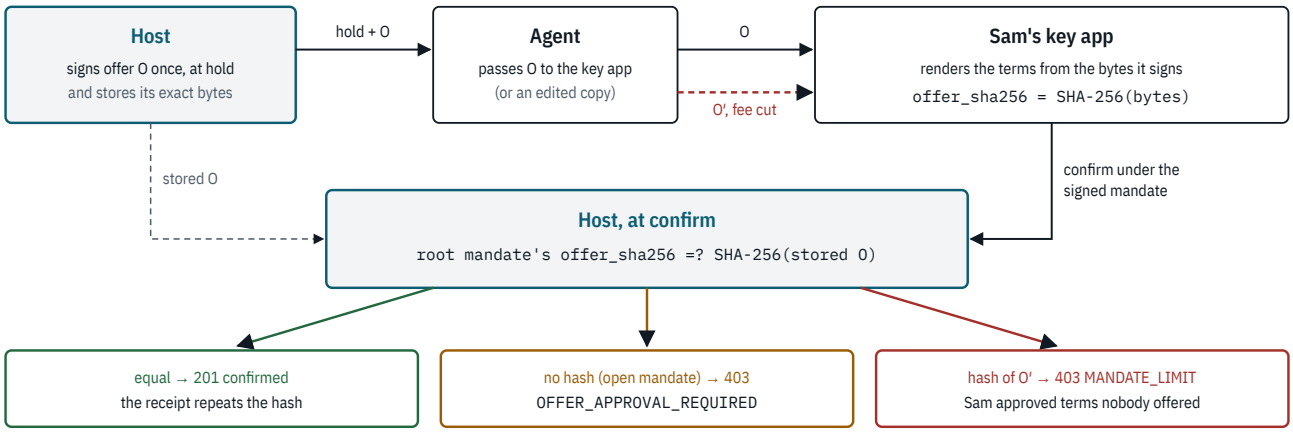



**Figure 14.** Closed mandates. The offer is signed once and stored, so its hash is fixed. The key app builds the consent text from the same bytes it hashes, so, provided the key app holds the key, the human can only approve terms the host actually offered. An agent that edits the offer before showing it gets a signature over a hash the host does not recognize. All three outcomes are tested in `test/trust.test.ts` and shown in the playground's signed-offer scenario.

The hold already carries the host's signed offer. A closed mandate is an ordinary mandate whose constraints include `offer_sha256`, the hash of that offer's exact bytes. It authorizes accepting that one offer and nothing else: a hold's offer, or a change offer. A host says when it insists through `policies.closed_mandate`: `never` or `always`, which then covers changes too. Harbourview Dental uses `always`. Up to the first v0.4 drafts a host could insist only when the offer stated a no-show fee; with fees gone from offers, that case went too. Agents read the same policy from discovery, so the planner asks the human at the confirmation step rather than after a refusal.

Two details carry the security. First, the hash must be in the *root* mandate, the one the principal's key signed. An agent-signed child carrying the hash proves only that the agent saw the offer, and the test for that case is refused. Second, the key app generates the consent text from the offer bytes it hashes. An agent that drops a term before showing it gets Sam's signature over different bytes, and the host rejects it. Every receipt names the contract a booking stands on and says how the person agreed: `consent.approved: "offer"` when the human's key signed the offer, `"limits"` when an open mandate covered it, and `"contract"` when they signed a meeting contract naming it, with the id of the mandate their key signed. The MCP tools state the

same fact from the user's side, as `user_signed`: "this exact offer", or "only the limits: service, time window, price ceiling". In live testing, assistants kept reading the field's earlier form, `approved_by: "agent"`, as "the user did not approve" (§10.21).

This rests on a trust boundary that no protocol check can see. A signature binds bytes, not what a screen showed. An agent that controlled both the display and the principal's key could show misleading prose over the original offer and still get a valid signature. The spec therefore requires one trusted component, the key app, to do three things: show the terms, generate the full text from the authenticated bytes, and hold the key. It must offer the agent no "sign this hash" call. The identity it shows is the domain verified during discovery. A key app with no discovery context labels the offer's claimed identity as unverified.

## 8.6 Relationship to AP2, and the next step

The reference implementation signs mandates as plain Ed25519 JWS, with the principal's public key embedded and a self-certifying identifier (`iss` = key thumbprint). That keeps the code legible and the tests deterministic. The intended evolution, recommended by our research, is to profile **AP2 v0.2** rather than keep a custom format. A first version of that profile is now built as an option beside the JWS format (spec §3.4, 11 tests):

- SD-JWT VC mandates with Key Binding and `vct` values such as `org.schcp.mandate.booking.open.1` (built);
- ES256 keys, which device secure enclaves and WebAuthn support (built for principals; request signing stays Ed25519);
- selective disclosure, so a mandate covering three garages reveals only the one being booked (built: each host sees its own entry and limits, and withholding one's own limits is refused);
- still to do: AP2's closed-mandate credential in place of the `offer_sha256` constraint of §8.5, but as a *stricter profile*, not a synonym. AP2 lets the agent close an open mandate with its endorsed key when the human is not present.[37] That shows the agent chose the terms, not that the human saw them. Where a host asks for the human's approval, SchCP's profile would require a closed mandate signed by the user's key over the host-signed offer.

The division of labour is now plain. A SchCP mandate authorizes time, never payment. If a host wants money, AP2 or ACP carries it, under their own mandates and beside SchCP's, not inside them. What SchCP defines that AP2 lacks are the scheduling constraint types, all built and tested:

- `services`;
- the time `window`;
- `max_price` applied to a booking's stated total;
- `max_bookings`;
- management scope (`bookings`);
- `offer_sha256`.

Constraints over recurring occurrences are not in v0.4. Two limits hold for any format and should be stated plainly. First, a booking limit shared across *several* hosts cannot be enforced offline; only the payment layer or an online issuer can enforce it. SchCP takes the second route: the key app's status service counts a claim per booking, a host that cannot get a claim refuses the confirm, and a cancelled booking gives its claim back. Second, money taken days after the purchase needs a stored card with the customer's agreement to merchant-initiated charges, which a one-time agent payment token does not provide.[14] That was one reason v0.3 recorded late outcomes instead of charging them, and v0.4 completes the move: SchCP neither charges nor records them, and a host that bills a no-show does it through its own payment arrangements, with the guest's agreement, outside SchCP.

# 9 Distribution and what hosts get

Protocols die without supply. For businesses, the original paper proposed three on-ramps, and the research changed how much each can carry. For people, the supply is their calendars, and the on-ramp is whoever already keeps them (§9.3).

## 9.1 Connectors to open APIs

Only a handful of platforms combine self-serve developer access, business-consented OAuth and a documented "search availability, then create booking" path: **Square, Acuity, Calendly, Cal.com and Microsoft Bookings**.[26] Even these come with caveats.

- **Square** requires a paid Appointments plan for full writes, and will not let an API book a service that carries a no-show fee. Those are exactly the businesses that care most about no-shows.
- **Mindbody** is public but metered, needs each studio owner to activate it, and charges integrations that aggregate across businesses $2.50 per appointment booking.
- **Vagaro**'s published reference has availability search but no create-appointment endpoint.
- **Jane** says outright that AI scheduling tools are "not being pursued" and that posting into its calendars is not supported.
- **Booksy** is partner-gated with partner keys, not business-consented OAuth. **Fresha** and **GlossGenius** have no booking API at all.

Most of these platforms have no hold primitive, so a connector must emulate holds locally. That works only if the connector is the sole writer, which it rarely is (§10.4).

## 9.2 The calendar fallback, recast as request-to-book

A calendar is not availability truth. Free/busy has no service menu, no durations, no capacity, no buffers and no resources. Walk-ins and phone bookings that never reach the calendar make it optimistic. Google's appointment schedules have no API, and Microsoft's availability calls exclude personal accounts.[26]

The reference implementation therefore treats a calendar-only business as *declared hours × declared capacity − calendar busy time*. Every booking goes to `pending_host` and is written back as a TENTATIVE event with an approval link. If the business doesn't answer within two hours, the booking lapses and the slot is released. It is slower than instant booking, but no booking happens without the owner's say-so. A calendar that has not been read is not a free one: such a host offers nothing before its first sync and nothing past the stretch it last read.

## 9.3 Hosted endpoints, for businesses and for people

For the long tail the endpoint has to be hosted for them, by a platform or by anyone already selling into small service businesses. On Census counts about nine in ten US beauty salon establishments have no employees, and NFIB reports a third of non-employer businesses have no website.[26] Our operator model (§5.1) is built for that case. Wix's USP and the UCP Local Services effort serve the same businesses, so an operator may want to speak both; SchCP's envelope is kept close to theirs for that reason.

People need the same thing, and the natural operators are the services that already keep their calendars. In the sample world, Daybook hosts one endpoint per person, each owned by that person's key app and open to their contacts only (§5.5). Nobody has to run a server, and the calendar stays where it is. Whether calendar providers want that role, and what it would cost them to carry it, is open (§11).

## 9.4 What businesses get, and what they fear

### Incentives

- Phone load. Phone is still the dominant channel.
- Demand capture. OpenTable reports AI-assistant channels drove 17× more seated diners year over year (company figure).
- Terms the customer's agent has verifiably shown the customer (the time, the place and any stated price), which an anonymous web form cannot offer.

### Objections

- No-shows and low-intent bookings: ~3–4% measured in salons (vendor data), ~23% across healthcare studies.[34]
- Marketplace fees: Booksy's optional Boost (30% of an eligible new client's first visit, capped at $100) and Fresha's one-time 20% on new marketplace clients.[31][32] Will an agent booking count as "marketplace"?
- Losing the customer relationship.

SchCP answers the first objection in part: mandates, bounded holds, terms the customer's agent has verifiably seen, and no-shows the host declares, which earn a guest no standing as a returning customer. It no longer offers deposits, or records fees for the host to bill. Whether businesses accept a protocol that never charges, and take deposits, if they want them, in their own checkout, is a question only pilots can answer (§11). It answers the third partly, because bookings carry the customer's contact and the receipt names the business. It cannot answer the second. Fees are a platform business decision, and they are the strongest argument for keeping the protocol free of any one platform.

## 9.5 Who does the work, who pays, and one operating example

A protocol moves bookings between parties who each carry a cost. The **operator** builds and maintains an adapter per booking platform (or, for people, per calendar provider), hosts the endpoint, runs the abuse controls and answers the host's support calls. The **host's system of record** (Square, Mindbody, a calendar) stays authoritative, and every adapter has to write into it without becoming a second ledger. The **assistant** carries the key app and the consent surface, and with them the liability for what the human was shown. There is no payment provider in the picture: a business that takes money does so through its own, outside SchCP, and a stored credential for any fee charged later is between it and the customer. SchCP standardizes how these parties talk. It does not pay any of them, and it says nothing about who earns what. Whether an agent-made booking is "marketplace" demand that carries a commission is a commercial question the receipts make answerable, not one the protocol settles.

The benefit to one business is easy to overstate. A no-show percentage multiplied by gross appointment value is not a saving caused by a protocol, so here is a smaller calculation, with every input labelled as hypothetical until measured:

| A TWO-BAY SHOP, PER YEAR | HYPOTHETICAL INPUT | VALUE |
|---|---|---|
| Scheduling calls | 50 a week × 3 minutes | 130 staff hours |
| Share handled by agents | half, at a loaded $25 an hour | ≈ $1,600 |
| Capacity filled that would otherwise sit empty | one $79 oil change a week, at a 40% contribution margin | ≈ $1,640 |
| Operator fee | $20 a month | – $240 |
| Before support and dispute costs | | ≈ $3,000 |

Two things follow, even with made-up inputs. First, the gross value of those filled slots ($79 × 52 ≈ $4,100) is not the benefit: only the margin is. Second, about $3,000 a year matters to a small shop but would never fund a custom integration. The economics only work if an operator builds each adapter once and spreads it across

many merchants, which is why §9.3's operator model carries the distribution argument. A pilot should measure the four inputs before anyone quotes an output.

# 10 Building it: tradeoffs and challenges

A protocol sketch hides its hardest decisions in phrases like "the host checks the mandate" and "holds expire." To find out what those phrases cost, we built SchCP end to end, then wrote tests meant to break it. This section reports what happened. The suite grew in passes: 77 scenarios from the first build, 97 after an independent review (§10.10), 108 after the second pass (§10.12), 155 after ten business-viewpoint reviews (§10.13), 163 once the four defects they found were fixed, 194 after a deep review reproduced 20 more defects, which we then fixed (§10.14), 216 after building v0.3 (§10.15), 243 after two reviews of v0.3 and the scenarios of §6 (§10.16), 292 after closing the reviewers' gaps and adding crash, conformance, MCP and AP2 tests (§10.17), 267 after v0.4 took the money out and added people as hosts (§10.18), 282 after meetings of several people and leaving outcomes to the host (§10.19, §10.20), 296 after live testing with AI assistants (§10.21), 297 with the conformance vectors for a guest's move, 321 after the open questions were decided and built (§10.22), and 330 once what a second implementation could not find in the text was written into the spec (§10.23). Of the 330, all pass except one, which documents a decision not to build arrival windows. Sections 10.10 to 10.17 describe earlier versions in the vocabulary of their time, when a host was a "merchant" and a guest a "customer".

## 10.0 What was built

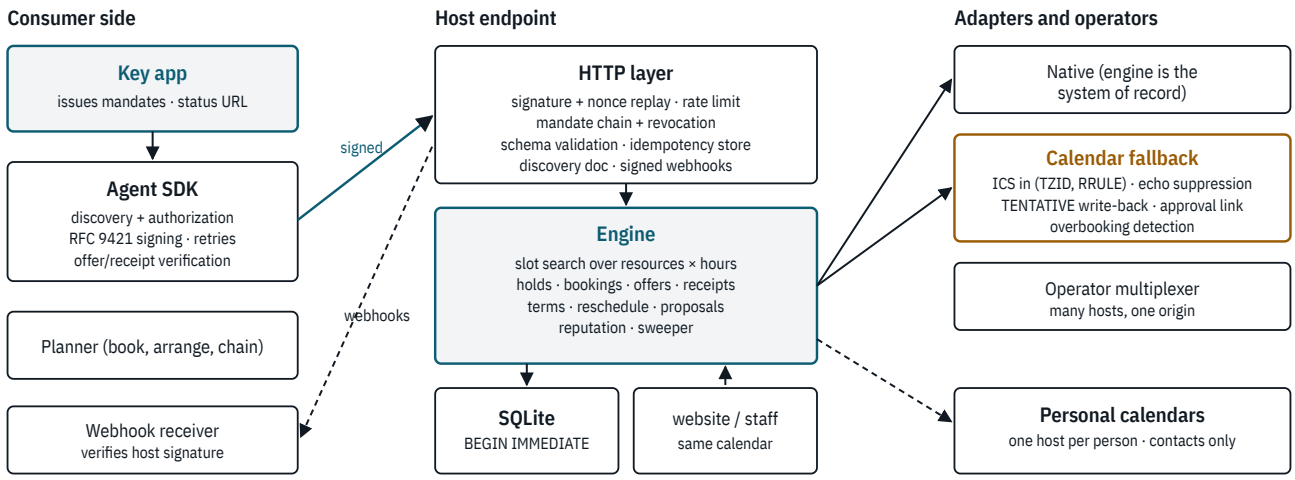



**Figure 15.** The reference implementation. TypeScript run directly by Node 22 (no build step) with four runtime dependencies: a JSON Schema validator, its format plugin, a timezone library and an iCalendar parser. Signing, mandates, idempotency and the engine are written from scratch so every rule in the spec has one place in the code.

| COMPONENT | WHERE | LINES |
|---|---|---|
| Protocol core: JOSE (EdDSA, ES256), RFC 9421 profile, mandates, SD-JWT VC, stated prices and totals, durations, errors, schemas | `src/protocol/` | ~2,400 |
| Host engine, HTTP layer, outbound fetch, store, calendar adapter | `src/host/` | ~4,900 |
| Agent SDK, key app, webhook receiver, planner (book; arrange, finish and revise a meeting of several people, before and after it is confirmed) | `src/agent/` | ~1,700 |
| Simulated internet: five businesses (one a café), a booking platform, a personal-calendar service with three people, DNS-like resolution | `src/sim/` | ~580 |
| MCP server for existing assistants: 16 tools | `src/mcp/` | ~1,700 |
| Conformance vectors generator and black-box runner | `conformance/` | ~2,400 |
| Spec and JSON Schemas | `spec/` | ~4,800 |
| Scenario, property, regression and conformance tests | `test/` | ~9,400 |

The test world is a small simulated internet. **Maple Street Auto** runs its own endpoint. **Fades on Queen**, a barbershop with no domain, is listed by a platform called Chairly. **Harbourview Dental**, in Vancouver, has a domain but books through Chairly. **Rosie's Nails** has only a shared calendar. **Bellwoods Café** holds tables for meetings of friends. **Noor Haddad**, **Ellis Moreau** and **Priya Shah** are three parents whose calendars live on **Daybook**, a personal-calendar service. Each calendar is owned by its person's key app and open to the other two as contacts. Noor and Priya have each signed a rule for the agent that keeps their calendar; Ellis has not. Every party signs and verifies against public `https://*.example` URLs while traffic is routed to local ports, so signatures exercise the real target-URI logic. A movable test clock lets holds lapse, mandates expire and signatures go stale on demand.

## Scenario coverage

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| Walkthrough | 4 | The oil change end to end; second and third guests at the same time (second bay, then a counter-offer); human declines at the last step; tampered offer | pass |

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| Holds and contention | 11 | Expiry; TTL caps; 10 agents racing for one barber; walk-in versus agent hold; fan-out; per-agent cap; anonymous holds; reputation tiers; key rotation; booking windows and closures; two-resource services | pass |
| Delegation | 13 | Price, currency, service, window and audience limits; stolen mandate; expiry; tampering; key substitution; revocation mid-flow; scope; sub-agent attenuation and widening; wildcards; cross-principal access | pass |
| Transport | 8 | Signature coverage; unsigned, mis-keyed and mis-targeted requests; replay; clock skew; lost-response retry; idempotency conflict; rate limiting; schema errors | pass |
| Lifecycle | 12 | Free and late cancellation, both recording nothing; a no-show the host marks, with no amount; reschedule (atomic, stale sequence, self-overlap, failure leaves original); a late change, free; host cancel with rebook; counter-proposals accepted, declined and lapsed; forged webhooks; management-mandate scope | pass |
| Calendar fallback | 5 | TZID without VTIMEZONE; recurring busy time; approval flow and webhooks; decline and timeout; a human writing into an agent-held slot | pass |
| Discovery | 8 | Own domain, operator-asserted and operator-mutual; lookalike spoof; unclaimed operator; hijacked listing; garbage documents; a service found by the words a person uses ("haircut and beard") | pass |
| Time | 6 | Duration parsing; the 1 November 2026 DST change in Toronto; a Toronto agent booking a Vancouver dentist; multi-person intersection; chaining two hosts; whole seconds on the wire from a clock that is not | pass |
| Property (fuzz) | 3 | 3 seeds × 12 agents × 30 rounds of random holds, confirms, releases, reschedules, cancels and refreshes, plus walk-ins and staff cancellations, with a moving clock. Invariants after every round: no double-booked resource, live bookings own exactly their allocations, no hold outlives its TTL, no mandate over its limit | pass |

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| Review regressions | 20 | One test per finding of the independent review (§10.10): approve/cancel race, hostile calendars, sybil holds, availability cost, proposal and reschedule state, revocation and quota keyed per principal, SSRF, echo suppression, ICS injection, scoped sync, concurrent confirms, malformed mandates, nonce expiry, resource choice, pruning, error classes, signature coverage | pass |
| Trust (v0.2) | 11 | Closed mandates: required by policy, one offer only, edited terms caught by the hash, agent-signed hash refused, planner asks only when needed. Attestations: verified newcomer gets in during a sybil flood; copied, impostor and expired attestations ignored; per-attester cap; attester loses standing. Decay: staged recovery from suspension; "has kept an appointment here" never decays | pass |
| Units | 3 | Money rounding and zero-decimal currencies; RFC 8037 thumbprint vector; schema-valid discovery documents | pass |
| Business seats (v0.5, closed in v0.9, re-read for v0.4) | 47 | Scenarios written from ten business viewpoints (§10.13). The 16 that marked gaps assert the behaviour the seats asked for (§10.17). Since v0.4 the payments-risk seat asserts that nothing can be charged: a request carrying payment data is refused, and the receipt is evidence of the stated terms, not of money taken | 46 pass<br>1 won't do |
| §6 scenarios | 5 | A week at a mechanic shop; a sales demo with a second decision-maker and reconfirmation; a meeting between colleagues, accepted, defaulted and declined | pass |
| People and open-ended bookings (v0.4) | 8 | A playdate on one contract across two contacts-only calendars, ended early; a stranger refused and a contact removed mid-hold; a named place signed into the offer; a waitlist that carries its place to the offered hold; a drop-off set aside until close and ended at 10:40; a no-show with no price; configuration rules | pass |

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| Meetings and answers (v0.4) | 14 | Three families on one contract hash, with a yes by person and a yes by rule, and a commit refused until everyone has answered; a contract that asks for the person, and one that asks it of one member only; a weekend rule that doesn't cover a Wednesday; a counter to another place and a revision everyone answers again, even when a member's yes to the old one races it; a yes to an earlier revision; a decline in the person's own words, a yes taken back, a member backing out and a lapse, each releasing every calendar; a meeting window that holds the whole meeting; each calendar's checks of its own entry; a rule's yes, a person's yes and a counter on an ordinary booking | pass |
| Guest moves (live testing) | 3 | At Rosie's Nails, which approves each booking: a move waits for her yes while the booking keeps its time; her no, and her silence, leave the booking where it was and tell the guest why; a rule Priya signed says yes to a move it covers, and one it does not cover waits for Priya | pass |
| Changing a confirmed meeting; a café as a member (v0.14) | 10 | A confirmed playdate moves only once everyone says yes, and every calendar keeps its time until then; a no withdraws the change everywhere and a lapse drops it, the meeting keeping its time; a change answered after the start, or with other members, refused; a calendar without the meetings capability refused; members shown by the reader's own contact names; Bellwoods Café as a member whose offer is its yes, one that approves each booking, and a later hour the friends do and do not agree to | pass |
| Records, receipts and shared limits (v0.14) | 9 | Retention in discovery, every offer and every receipt, a change offer keeping the booking's own; receipts saying how the person agreed and under which mandate, and the SDK refusing one that disagrees; the MCP pointing out what differs from the request; a cancel giving a unit back to a shared booking limit, once only, with the key app down, after a day without an answer, and on a lapse | pass |

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| What the spec now states (v0.15) | 9 | A repeated JSON member refused anywhere, in a body or a signed payload; host signatures named by `kid` and only that key used; price, total and lines together and summing; a fee inside `slot` refused like one at the top; exact percentages; retention in weeks or days; a child that keeps the principal, whose wider limit the parent still refuses; an expired mandate named as such; an agent key carrying a private member refused | pass |
| v0.3 reviews | 18 | Tests for the defects of the two v0.3 reviews (§10.16) that still apply after v0.4: waitlist and hold controls, tentative bookings, contract terms and reconfirmation, replay, receipts, calendar overrides, tooling | pass |
| Deep review (v0.6) | 22 | Replay, read and release authorization, receipts in the SDK, connect-time URL checks, revocation chains; signed terms across policy change and restart; recurrence overrides, buffers, write-back, wire shapes; standing from kept appointments (§10.14) | pass |
| Conformance | 27 | Wire-level vectors checked against a fresh build and against `node:crypto`, including a guest's move read from receipts and webhooks and the revisions of a meeting, before and after it is confirmed; the black-box runner against sample hosts and deliberately broken endpoints; every hold, waitlist entry and booking response the endpoint returns checked against its closed schema, and one carrying a field SchCP does not define refused | pass |
| Consent text | 6 | Local time in words with the host's name and verified domain; an unverified claim flagged; a change's terms; an unpriced, open-ended meeting at a named place; a reconfirmation, with its deadline and what silence does; the host's records retention in plain words | pass |

| AREA | TESTS | WHAT IS PUT THROUGH IT | RESULT |
|---|---|---|---|
| MCP server | 15 | Handshake and versions; bookings at Maple (open mandate) and Harbourview (the human signs the offer), a waitlist at Fades and a freed time taken from it, a playdate on a contacts-only calendar, and a meeting of three families with a counter and a revision, plus a rule's yes and a counter on ordinary bookings, a move at Rosie's that waits for her answer, a haircut with a chosen barber, and a confirmed meeting changed with every member's yes, members shown by the user's own contact names, through tool calls; refusals mapped to tool errors; no key or mandate ever in the model's view | pass |
| AP2 profile | 11 | RFC known-answer vectors for ES256, thumbprints and SD-JWT; Key Binding failures; selective disclosure across three hosts; open and closed SD-JWT mandates booking end to end | pass |
| v0.3 | 22 | Contract hashes on every acceptance and the change chain; a host-signed offer with a term SchCP doesn't define, refused before the human sees it; a late cancellation that records nothing and carries no balance to the next offer; refusal reasons and contact remedies; cancelling keeps a booking's reputation credit; the waitlist (first come, same chair, lapse moves down the line, windows that must fit, and one too short for the appointment refused when joining); tentative bookings, participant answers, defaults and reconfirmation (§10.15) | pass |

## 10.1 Atomic allocation is the whole game

"Holds prevent double-booking" is only true if the check and the write are one indivisible step. The engine models availability as *resources* (bay 1, Alex the technician, chair 3), each with working hours. A service requires one resource of each listed type for its duration plus buffer. Holds, bookings, pending approvals, host proposals and imported calendar events are all rows in one `allocations` table. Every critical section runs inside a single `BEGIN IMMEDIATE` transaction with no `await` inside it: the store throws if a callback returns a promise.

Within one process this gives serial execution, and across processes sharing the file it gives SQLite's write lock. The race test sends ten concurrent holds for one barber and gets exactly one winner and nine counter-offers. The fuzz test's no-overlap invariant held across about 1,100 random operations. A production deployment on Postgres would want an exclusion constraint over time ranges as a second line of defense. Here we chose the simplest primitive that is obviously correct.

That paragraph was true and still misleading. Allocation was atomic inside each transaction, but a confirm with a deposit is not one transaction (§10.2). The deep review found that a lease which ran out during a payment call could be revived after another customer's hold had taken the slot. "Atomic" has to hold across the whole saga, not just each step of it. v0.4 removed the saga: with no payment call, confirm is one transaction again (§10.18).

## 10.2 Payments break the transaction

History. v0.4 removed deposits, and this saga with them (§10.18). The section stays because the failure class it describes is the main reason money left the protocol.

Deposits introduce an `await` in the middle of a confirm: the call that captures money at the payment provider. That call cannot sit inside the database transaction. So confirm runs in three phases.

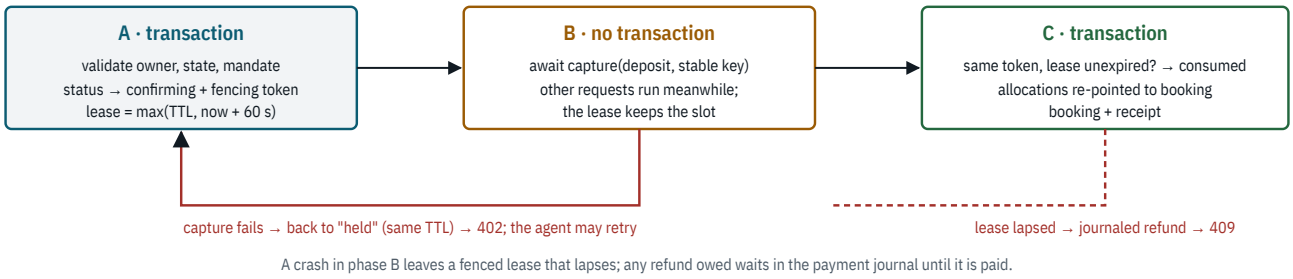



**Figure 16.** A saga inside the endpoint. AdCP's spec makes the same distinction between an "atomic mutation" and "a saga, not one atomic transaction."[1] The capture-failure path is a test: the hold survives, and a retry succeeds. So are a lost capture response, a lease that lapses mid-capture, and a refund interrupted by a restart.

The first version of this saga had three gaps, all at the edges of phase B, and the deep review found each one (§10.14). v0.6 closes them. Phase A tags the hold with a fencing token, and phase C succeeds only if the token still matches and the lease has not run out, whether or not the sweeper has noticed. The capture carries a stable idempotency key, and a timeout is an unknown outcome to reconcile with the provider (`PAYMENT_UNKNOWN`), not a decline. Every refund or void is written to a payment journal before it is attempted, so a crash between losing the race and paying the money back is finished on restart. One edge remained for a test that kills real processes to find: a capture sent by a merchant that died can land while its successor is voiding the same authorization (§10.17).

## 10.3 Mandate lifetime ≠ booking lifetime

> **FOUND BY THE TESTS**
> Our first lifecycle tests booked Thursday's appointment on Tuesday, then moved the clock to Thursday morning to cancel. All three failed with `MANDATE_INVALID: mandate expired`.

The design was internally consistent and still wrong. Short buying mandates are a security property, since a leaked mandate should be worthless within the hour. But the human's need to cancel or reschedule lasts as long as the booking. Stretching the buying mandate to Thursday would have kept a `hold` / `book` credential alive for two days.

We split the two. The buying mandate stays short. At confirmation the key app issues a management mandate scoped to that booking ID, to `cancel` / `reschedule` only, valid until the appointment ends (§8.3). The engine enforces that a mandate carrying `bookings` can do nothing else. A test checks that it cannot place holds or touch other bookings. The same gap exists in any "one token per purchase" design, including payment tokens that are `one_time` by definition.

## 10.4 Holds on top of systems that have none

The calendar adapter was where real-world mess showed up.

- **The floating-time trap.** A Google or Outlook ICS event reads `DTSTART;TZID=America/Toronto:20261015T130000`. If the feed omits the `VTIMEZONE` block, the standard JavaScript iCalendar library treats it as *floating* time and converts it in the *server's* local zone. On a developer laptop in Toronto it is correct. On a UTC cloud host it is four hours off, and every busy block lands in the wrong place. The adapter now resolves `TZID` through the IANA database itself, and a test pins a Vancouver event to 20:00Z regardless of where the test runs.
- **Echo suppression.** Our own bookings are written back to the business's calendar so the human can see them. On the next sync they come back as busy time and would be counted twice. Events with SchCP's UID suffix are skipped.
- **Recurring busy time** (the lunch break) has to be expanded, not read as one event. `TRANSPARENT` events must not block anything.
- **Two writers race.** In the test, Rosie types two regulars into 4 PM while an agent holds a 4 PM station. The adapter detects the overbooking on sync, and the booking still lands in `pending_host`, where Rosie sees the conflict before approving. A connector to a platform without holds would have the same problem with no approval step to save it. That is why §9 recasts the fallback as request-to-book.

## 10.5 Reschedule overlaps itself

Moving a 3:00–4:00 haircut to 3:30 with the only barber looks impossible to a naive availability check, because the barber is "busy" from 3:00 to 4:00 with this very booking. Releasing first and then allocating is not safe either: if the new slot fails, the customer has lost the original. The engine renames the booking's allocations to `…:old` inside the transaction, searches with those ignored, allocates the new slot, and only then retires the old one. Any failure rolls the whole thing back. Two tests cover this: the self-overlap move succeeds, and a failed move leaves the original booking and its `sequence` untouched.

## 10.6 Nonces and idempotency keys are different things

It is tempting to use one value for both. They answer different questions. The signature nonce answers "has this exact message been seen before?" and must be fresh on every attempt, or a legitimate retry looks like a replay. The idempotency key answers "is this the same *intent* as before?" and must be *stable* across retries. The SDK re-signs each attempt with a new nonce and keeps the body, including its idempotency key, byte-identical. A test drops the response to a successful confirm on the floor. The SDK retries and gets the stored response, flagged `Idempotent-Replayed`, and exactly one booking exists. Deterministic errors (a 409 for a taken slot) are stored as well. Rate limits and server errors are not, so they can be retried.

## 10.7 What you sign is the public URL

RFC 9421's `@target-uri` is the URL the client meant, but behind a TLS terminator or a platform's router the server sees something else. Each of our operators multiplexes two hosts under one origin, and every test dials a local port while signing `https://*.example`. The endpoint reconstructs the target from its configured public origin, never from the `Host` header. A test signs for the internal address and is correctly rejected. Implementations will get this wrong. The spec should name the rule.

## 10.8 Reputation keyed on agent keys is theatre

The first reputation design tracked agent keys only. Minting a new Ed25519 key takes microseconds, so an abusive operator simply rotates. Tracking the *principal* as well closes that loop, and a test rotates keys on every hold and is still suspended. It moves the problem to "how expensive is a principal?" The review answered that one: in v0.1, a principal costs nothing either (§7.2). Reputation is necessary but cannot carry the load alone. The v0.2 attestations (§7.5) are a first answer that needs no central registry.

## 10.9 Small decisions that mattered

- **Money as decimal strings, minor units in memory.** 25% of $79.00 is $19.75 exactly, and 33% of $0.10 rounds half-up to $0.03. Zero-decimal currencies come from `Intl`. A USD limit never approves a CAD price.

- **Only exact durations.** `P1M` is refused because "a month" has no fixed length. Booking windows are written as `P60D`.
- **Offsets on every timestamp, plus an IANA zone in the document.** The DST test asks for a window in UTC across the 1 November 2026 change and gets back Saturday 09:00-04:00 and Monday 08:00-05:00: local opening hours, correct offsets.
- **Alternatives filtered agent-side.** The host's counter-offer may fall outside the human's window. The planner discards those rather than "helpfully" booking them, so the host is never asked to enforce the agent's limits.
- **Fees without a payment instrument were only ever a record.** In v0.2, with no captured deposit, a late-cancellation fee was written as `fees_charged` and could not be collected. v0.3 made that the rule rather than the exception, and v0.4 the whole rule: SchCP collects nothing, and since §10.20 it records nothing either (§8.2).

## 10.10 What an independent review found

After the suite passed, we gave a separate agent the code and one instruction: break it. It had no part in writing the code. It wrote 17 proof-of-concept scripts against the running endpoint and found 18 problems. We fixed all 18, re-ran its scripts against the fixed code to confirm each is closed, and added one regression test per finding. The suite went from 77 to 97 tests.

| # | FINDING | SEVERITY | FIX |
|---|---|---|---|
| 1 | Merchant approval raced a customer cancel: the booking became "confirmed" with no slot behind it, the slot was resold, and the cancelled customer's deposit stayed captured | high | The same lease → pay → conditional-write saga as confirm (§10.2); refund on a lost race |
| 2 | A recurring calendar event with a Windows zone name (`Eastern Standard Time`) froze the whole server in an infinite loop; one-off events with such names were silently dropped | high | Resolve TZID via IANA, then the feed's VTIMEZONE, then known Windows names; check every value is finite; cap recurrence expansion; return warnings |
| 3 | Sybil squatting: fresh identities per hold bypassed every per-identity control; one script held a whole day | high | Cross-identity cap on holds by unproven principals; late release counts as expiry; spec corrected (§7.2) |

| # | FINDING | SEVERITY | FIX |
|---|---|---|---|
| 4 | One availability query cost ~0.5 s of blocked event loop; fresh keys bypassed the rate limit | high | Working hours computed once per query, not per candidate (536 → ~30 ms); at most 3,000 candidate starts per query |
| 5 | Accepting a proposal on a cancelled booking blocked the slot forever | medium | State checks; every exit from "confirmed" clears the open proposal and its slot |
| 6 | Rescheduling after the start time dodged late and no-show fees | medium | No reschedule after start; negative notice gets the strictest tier |
| 7 | A revoked mandate could be revived by minting another principal's mandate with the same `jti`; the same id let a stranger use up a victim's `max_bookings` | medium | Revocation cache and usage keyed by (principal, jti) |
| 8 | SSRF: the endpoint fetched attacker-chosen status and callback URLs, including internal admin paths, with no timeout | medium | https-only URL guard (no private, loopback or link-local addresses, no credentials), timeouts, no redirects |
| 9 | Calendar echo suppression ignored *any* `.schcp` UID, so the owner's own appointment booked elsewhere over SchCP didn't block time | medium | Suppress only exact echoes of our own live bookings at their current times |
| 10 | A customer's name could inject iCalendar lines into the merchant's calendar | medium | RFC 5545 TEXT escaping |
| 11 | Syncing one calendar wiped another resource type's busy time | medium | Sync scoped to its resource type |
| 12 | Concurrent confirms could exceed `max_bookings` | medium | Mandate usage recorded in phase A, before payment |
| 13 | Malformed constraints failed open: an unparseable window bound was ignored, and a string `services` matched by substring | low | Mandate claims validated against a schema; malformed means invalid |
| 14 | A one-second replay window at the edge of the signature's freshness window | low | Nonces kept until `created + skew + 1 s` |
| 15 | Rescheduling dropped the customer's chosen barber; a second late fee overwrote the first | low | Chosen resource persisted; fees accumulate |

| # | FINDING | SEVERITY | FIX |
|---|---|---|---|
| 16 | Unbounded state: rate-limit buckets, idempotency rows, webhook dedupe set | low | Periodic pruning; bounded sets; body size limits |
| 17 | Empty durations and wrong-currency fees produced 500s; fractional granularities offered unbookable starts; a month spanning DST was rejected | low | Stricter schema; typed 400s; whole-minute granularity; DST slack |
| 18 | Hardening: the signature didn't cover the mandate header; operator discovery trusted the operator's copy; redirects followed; management mandates could release holds; anonymous holds could never be confirmed | low | Mandate header signed; both documents must agree; no redirects; scope checks on hold operations; anonymous holds claimable by the same agent |

Four lessons stand out.

1. **The paper had a false claim, and only an adversary found it.** "Every hold costs a human signature" was the backbone of §7 in the first draft. It is true only if principals are expensive, and v0.1's are not. No test we wrote could have found this, because our tests used one key app per simulated person. The reviewer simply did not.
2. **Every pattern must be applied everywhere.** We had designed the lease → pay → conditional-write saga carefully for *confirm*, then wrote *approve*, the same shape on the merchant's side, without it. Protocols have more entry points into the same state than their authors remember.
3. **Identifiers chosen by the caller are not keys.** A `jti` is picked by whoever signs the mandate. Using it alone as a cache key let a stranger overwrite a victim's revocation status. Anything a caller names must be namespaced by something the caller cannot choose.
4. **A protocol that makes servers fetch URLs is an SSRF protocol** unless it says otherwise. Status endpoints and webhooks are both "please call this URL." The spec now carries the URL policy as a MUST.

The review also checked properties that held up: atomic allocation under concurrency, the confirm lease, reschedule rollback, mandate-chain binding, the signature profile's strict parsing, idempotency conflict detection, deposit-reuse prevention, discovery's subdomain rule and webhook deduplication.

## 10.11 The sample booking site

Tests prove behaviour, but they don't show it. The repository includes a sample site (`npm run site`) that runs the same simulated internet with a real clock. Every action on it is a real, signed SchCP call through the SDK.

- **Public pages.** Each business has one with services, hours, policies and its discovery document, plus a booking widget for humans. The widget draws on the same availability agents see, so a slot booked there disappears from agents' availability.
- **Staff dashboards.** A per-resource timeline shows holds counting down beside confirmed bookings, pending approvals, host proposals and imported calendar events. Staff can approve, decline, cancel, propose a new time, or mark a no-show, and a live log streams the protocol events.
- **Agent playground.** It runs nine scenarios against the live endpoints and shows each one message by message: the walkthrough, a barber race, hold-and-walk-away, a price above the mandate's limit, a cancellation (in v0.3 a late one that was undone or recorded; since §10.20 simply free), a calendar-only business's approval, a spoofed endpoint, and two added with v0.2: a dentist that requires a signed offer, and a sybil flood that a verified newcomer gets through.

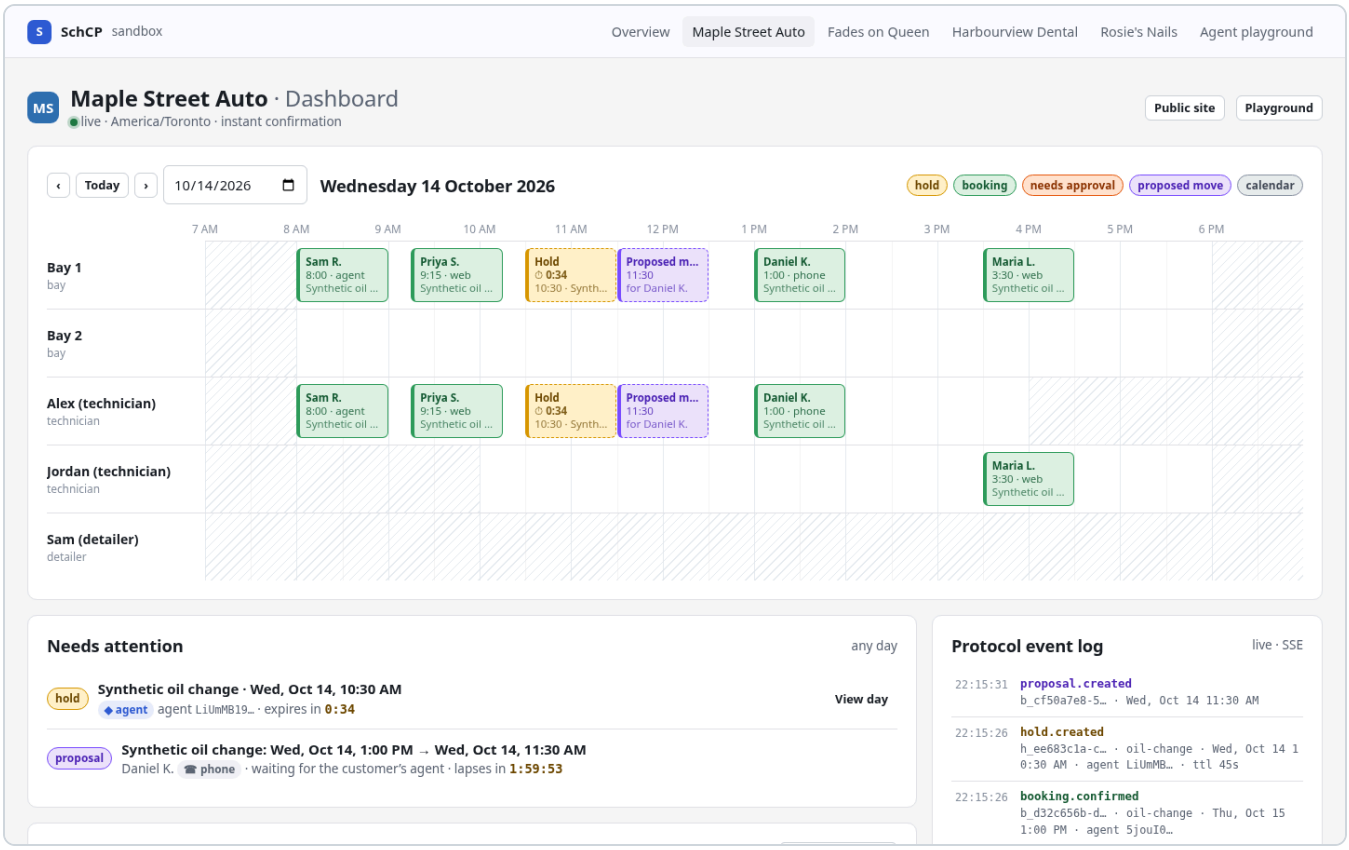



**Figure 17.** The host's side, at a business. Bookings from agents, the website and the phone sit in one timeline because they share one calendar. The amber block is an agent's hold with 34 seconds left. If it lapses, the slot returns to every channel at once. The purple block is a counter-proposal waiting on the guest's agent. (The screenshot is from v0.9; v0.4 changed the labels, not the layout.)

Building a human interface on top of the protocol surfaced three more issues that the API-level tests had missed:

- Marking a no-show charged nothing, although the policy and the spec both say it should. It then applied the no-show fee and kept it from the deposit. (v0.3 later replaced charging with recording; §8.2.)
- A merchant's "decline" offered the declined time back as the first alternative. It is now excluded.
- The merchant's own website books instantly even at a calendar merchant that requires approval for agents. We kept that, because the merchant's own channels are the merchant's choice and the approval step exists to protect them from agents. It is now written down rather than accidental.

## 10.12 Second pass: building the open questions

The third draft of this paper listed closed mandates, trust in principals and reputation decay as open questions. All three could be built without changing the shape of the protocol. Each adds an optional field and a merchant policy, and a merchant that ignores them behaves as before. The spec is now v0.2, 11 new tests cover the additions, and two new playground scenarios show them. Each still had a trap in it.

- **Naive decay never forgives.** The obvious design multiplies every count by the same factor over time. But uniform decay leaves the expiry *ratio* unchanged, and a suspended agent cannot place holds, so it can never earn the good outcomes that would lower it. A half-life alone keeps the suspension permanent. The fix is a floor: divide by at least `min_sample`, so that as old expiries fade they stop counting as enough evidence on their own.
- **We got our own numbers wrong.** The first draft of the v0.2 spec said a suspended agent recovers fully "after three weeks." The test written to pin that sentence down showed one week, because the tiers are crossed sooner than intuition suggests. The spec was corrected, and the test now checks every step: 60 s after a day, 120 s after three days, full after a week.
- **An offer must be an artifact, not a view.** v0.1 signed the offer afresh whenever a hold was rendered. The offer includes the hold's expiry, which the confirm lease changes, so the "same" offer could have two hashes. Closed mandates need fixed bytes, so the endpoint now signs the offer once, at hold time, and stores it.
- **The consent text is part of the protocol.** The first generated consent text read "cancelling with less than PT0S notice costs CAD 75.00". That describes the dentist's tiers backwards: the fee applies with less than 48 hours' notice. No assertion caught it; reading the playground output did. A human signs that sentence, so a test now checks it.
- **A test that passes first time proves little.** All 11 new tests passed on their first run, so we broke the code on purpose: no decay, no closed-mandate check, no separate attester pool, attesters that never lose standing. Each change should have failed a test. One didn't. The attester-pool test still passed with the feature removed, because Fades on Queen's two caps were both six. The test now sets them apart.
- **Lanes, not gates.** Attestations could have been a gate ("only attested principals may hold"), which would make the attester a toll booth. They are a separate lane instead, chosen by each merchant, that falls back to v0.1 behaviour when an attester misbehaves. Closed mandates follow the same logic: they are merchant policy, not a protocol requirement, because a free-cancellation haircut with no deposit doesn't need one.

## 10.13 Third pass: ten seats at the table

Every test so far had been written from one seat: the protocol designer's. For the third pass we asked ten reviewers to read the spec, code and paper each from a single business viewpoint and write a report. Nine also wrote runnable scenarios against the reference endpoint. A scenario that passes shows something v0.2 already handles; a *todo* scenario marks a gap, and each one fails on the gap assertion itself rather than on setup, so it flips to a pass when the gap is closed. The reviews produced 47 scenarios, 26 of them gaps. The suite went from 108 to 155 tests. Fixing the defects below added eight regression tests and turned four gap scenarios into passes, leaving 163 tests: 141 passing, 22 todo.

| SEAT | PASS / TODO AT V0.5 | VERDICT |
|---|---|---|
| Small salon or barbershop owner | 3 / 2 | Not yet, but close |
| Booking-platform incumbent | 3 / 2 | Yes, early and defensively, if fleet-wide controls land |
| Healthcare clinic | 2 / 3 | Conditional yes for the booking front door |
| AI-assistant vendor | 3 / 2 | Pilot, not the default path |
| Payments risk and fraud | 2 / 3 | Underwrite deposits at confirm; not no-show fees on file |
| Multi-location chain | 2 / 4 | Pilot locations now; chain-wide later |
| Home and field services | 2 / 3 | Fixed-price visits yes; dispatch no |
| Consumer-protection regulator | 2 / 4 | Five requirements before scale |
| Scarce slots (restaurants, classes) | 2 / 3 | Off-peak yes; the 10:00 drop no |
| Investor and standards strategist | n/a | Strong engineering, unpersuasive paper |

Historical counts. After the deep-review fixes and v0.3: payments risk and the regulator are 5 / 0, healthcare and scarce slots 3 / 2, home services 2 / 2 plus one won't do; `reviews/README.md` keeps both columns.

Most of what the seats found are missing features. They were listed in §11, and v0.9 builds every one that had a scenario (§10.17). Four were bugs: things v0.2 claimed to do and did not. A 108-test suite had missed all four, because every one of those tests asked the designer's question ("does the protocol enforce its rules?")

and none asked the issuer's, the regulator's or the merchant's. All four are fixed in this version, each with a regression test. The no-show contest window the payments seat asked for is specified as future work, not built.

| # | DEFECT | SEAT | FIX |
|---|---|---|---|
| 1 | A deposit authorization was checked for status, merchant and amount, but not for whose it was. A leaked `authorization_id` could secure someone else's booking | Payments risk | The authorization must belong to the principal placing the hold; otherwise `PAYMENT_DECLINED` |
| 2 | Refund failures were swallowed on merchant cancellation and late cancellation, so a booking could say "refunded" when no money moved: a "credit not processed" chargeback the merchant loses | Payments risk | The failure is recorded on the booking, retried with backoff, and exposed to both sides. Every retry carries the same idempotency key at the rail, because a refund whose response was lost looks exactly like one that failed, and a naive retry would pay the customer twice. (The deep review later found these retries did not survive a restart; §10.14.) |
| 3 | Marking a no-show was silent: no sequence bump, no event, so the customer's agent never learned a fee had been kept | Payments risk | A no-show bumps the sequence, emits an event and records the fee on the booking |
| 4 | The consent text a human signs printed raw ISO timestamps and an internal identifier instead of the business | Regulator | The signed offer carries the merchant's name, time zone and domain. The key app shows the domain it verified during discovery rather than the one the offer claims, so a merchant cannot borrow another's. Since v0.6 it labels a claimed domain as unverified when it has no discovery context. The text reads "Cleaning and check-up at Harbourview Dental (harbourviewdental.example), Thursday 15 October 2026, 8:00–9:00 a.m. Pacific time, CAD 180.00; cancelling with less than 48 h notice costs CAD 75.00; …" |

Four lessons.

1. **The designer's seat is one seat.** The independent review in §10.10 attacked the protocol on its own terms. The business reviews asked whether anyone would run it, and found bugs the adversary had not, because the adversary and the designer share a frame. Each seat is a separate test oracle.
2. **v0.2 is built for a merchant you already chose; agents shop.** Its mandates, holds and abuse controls are scoped to one merchant. An assistant asked for "a haircut near me on Saturday" wants one booking limit across many shops; a platform hosting 4,000 salons needs an agent suspended at one to be suspended at all of them. The largest cluster of gaps is here.
3. **Money needs the issuer's view, not just the protocol's.** The protocol never moves money, and we had read that as "money is someone else's problem." The issuer's question is different: who authorized this, where is the evidence, and did the refund actually happen. Defects 1 to 3 all live in that gap. (v0.4's answer is to leave money out entirely; §10.18.)
4. **Consent text is UI, and UI was untested.** §10.12 found the consent text backwards; this pass found it unreadable. A sentence a human signs is the protocol's user interface, and it needs tests written from the reader's side, not the parser's.

## 10.14 Fourth pass: a deep review

For the fourth pass, a reviewer read the brief, spec, code and paper together and checked five things: agreement between them, security, external facts, persuasion and test quality. It wrote 21 probes, each a script that *passes* when it reproduces a defect. It found 20 defect categories: nine P1 ("fix before relying on the guarantee") and eleven P2. None was a missing feature. Each was something the paper, spec or code said already worked.

We fixed all 20, or so we thought: a later review found remaining cases in five of them (§10.16). Each probe was then turned around into a regression test that asserts the correct behaviour, and each test was checked by breaking its fix on purpose. Two such breaks went uncaught: a fallback that applies only to offers signed before the change, and a second layer of shape checks behind an error boundary that is tested. 20 of the 21 probes now fail. The remaining one checks only the textual URL pre-filter, which by design cannot see where a hostname resolves; the enforcement it stood in for now happens at connect time. The suite went from 163 to 194 tests.

| GROUP | WHAT WAS WRONG | FIX |
|---|---|---|
| Money and leases<br>4 × P1 | A lost capture response was read as a decline, so money was taken with no booking. Refunds owed by a confirm that lost its race could vanish if the process stopped. Two concurrent merchant approvals could refund the winner. A lease that ran out during capture could be revived over another customer's hold. | Captures get a stable key at the rail, recorded through the hold's state; every refund and release gets a durable journal entry, keyed for the rail, before it is attempted. An unknown outcome is reconciled with the provider (`PAYMENT_UNKNOWN`). Each confirm attempt carries a fencing token, approvals are single-flight, and an expired lease is never revived. |
| Authorization<br>3 × P1, 3 × P2 | A credential scoped to one booking could read another. An idempotent replay returned a stored result to a different principal. The SDK verified a receipt but returned unsigned data next to it. The URL guard checked hostnames, not the address connected to. Releasing a hold ignored its service and time. A child mandate could fill its root's revocation cache. | Reads, quotes and releases are checked against every link's constraints. Replays are authorized like fresh requests. The SDK returns the receipt's copy of the booking. Outbound fetches check the resolved address inside the socket's own DNS lookup. Revocation is checked root first and cached per link. |
| Accepted terms<br>P1, P2 | Fees came from the merchant's current policy, not from the offer the human signed, so a policy change or a restart changed existing bookings. An omitted reschedule policy made late moves free. | The booking stores the accepted offer's terms. Reschedule terms are in the offer, and default to the cancellation tiers. |
| Abuse<br>P1 | Booking and cancelling, both free, turned a throwaway identity into a "returning customer" outside the shared new-customer pool. | Standing is earned only by a kept appointment (§7.2). |
| Calendar and wire<br>7 × P2 | Buffers ran past closing time. Lapsed approvals stayed on the merchant's calendar. Recurrence exceptions were ignored. Old series and expansion limits read as free time. Malformed attestations threw errors. JSCalendar output lacked `updated`. Three webhook types had no receipt. | Buffers must fit working hours. Failed write-backs are retried. `RECURRENCE-ID` overrides are applied. An incomplete expansion keeps the previous busy time. Attestation parsing is contained. `updated` is emitted. Every booking event carries a receipt. |

The same review audited the paper's external claims. §1, §4, §8.6, §9 and §11 carry its corrections, and the sources now link to the pages that were checked. It also listed smaller disagreements between paper, spec and code: clock skew, versions, completion semantics, the consent trust boundary and credential renewal. The spec now settles each one.

Five lessons.

1. **Correct mechanisms fail where they compose.** Every P1 sat at a boundary: an `await` in the middle of a saga, a process restart, a credential delegated twice, a policy edited after signing. Each mechanism had its own tests; the joins between them had none.
2. **A lost response is a third outcome.** A payment call does not only succeed or fail. A timeout means "unknown", and treating it as either one loses money. PAP's operations draft makes the same rule for its own actions.[39]
3. **What was signed must be what is enforced.** The closed mandate proved Sam had seen a fee. The engine then charged whatever the policy said on the day of the cancellation. A signature over terms is only worth the code that reads them back.
4. **Free actions cannot earn trust.** v0.5 promoted any principal with a confirmed booking. Confirming and cancelling cost nothing, so standing cost nothing. Trust has to follow something an attacker pays for: here, turning up.
5. **Our tests could not see these boundaries.** The fuzz test never changed identities, never let an appointment end, never used deposits and never lost a response, which is why it found none of these defects. The reviewer also recommended work this pass did not do. One item is wire-level vectors that an independent implementation can run. The other is crash tests that kill a real process, rather than reopening its database in a new engine.

## 10.15 Fifth pass: building v0.3

The open questions in the previous version of this paper had become design decisions. Protocol v0.3 builds them:

- every agreement is a merchant-signed offer accepted by its hash (§5.4);
- offers state an all-in total;
- late cancellations, late changes and no-shows are recorded rather than charged (§8.2);
- a 60-second undo, a waitlist and two-party confirmation (§7.6);
- one refusal shape with registered reasons, every one offering a way to reach the merchant without an agent.

The suite went from 194 to 216 tests. Five gap scenarios became passing tests: the regulator's undo, all-in total and refusal reasons, the scarce-slot waitlist and the guardian booking. The home-service arrival window became an explicit "won't do": bookings are for exact times.

Five lessons.

1. **A hash can only name what exists.** "Every acceptance carries the offer's hash" sounds like a field. For a reschedule it changed the operation: an agent cannot hash an offer the merchant has not signed yet. So a change is now two steps, a change offer that holds the new time and then its acceptance. Merchant proposals already had that shape, and now they are the same thing.
2. **Recording is a more honest default than charging.** v0.2's most elaborate flow was the late-cancellation escalation: quote, refusal, a second refusal, a fresh mandate for one fee. The fee was usually uncollectable anyway, because nothing on file could pay it. Recording the outcome and putting it on the next offer, as a taxed line the human approves with their own key, removed that flow, a mandate constraint and a class of disputes. It also moved a cost onto merchants, who may want the money now. That is the first thing a pilot should test (§11).
3. **Everything that locks a time slot is a hold.** The undo, a waitlist offer and a tentative booking each looked like a convenience, and each was a new way to lock a slot for free. We made each count against caps and reputation the way a hold does, or thought we had: the waitlist and tentative bookings still slipped past several controls (§10.16).
4. **Tests share more than you think.** Two new tests changed a merchant's policy, and a later test in the same process saw the change. The simulated world copied each merchant's configuration only one level deep. It now copies it fully. No earlier test had changed a policy in place.
5. **Breaking it on purpose still pays.** We broke each new rule 40 times in all and checked a test failed. Two breaks went uncaught at first: a waitlist offer for an appointment that ran past the waiter's window, and a booker declining their own reconfirmation. Both now have tests.

## 10.16 Sixth pass: two reviews of v0.3

Two reviews of v0.3 ran in parallel, each without seeing the other: one by a separate agent, one in a separate session. Each wrote probes that pass when they reproduce a defect. Between them they reproduced 22 distinct defects, and they overlapped on only four. The second also checked the first deep review's 20 fixes. Five had remaining cases: a superseded approval could still refund a deposit (D07); refunds owed by an older database were never adopted into the journal (D06); replays still skipped service and price limits (D03); two contract terms were still outside the signed offer (D01); and a moved recurrence could still vanish (D14). §10.14's "fixed all 20" was premature. All 22 are now fixed. Each has a regression test that asserts the required behaviour, and each fix was checked by reverting it.

| AREA | WHAT WAS WRONG | FIX |
|---|---|---|
| Waitlist | A waitlist offer skipped every hold control: reputation suspension, the per-agent cap, the newcomer pools and the TTL cap. A few throwaway entries could keep a freed slot off the market for an hour and a half. A merchant's "the lift is broken" cancellation offered that same bay to the next customer. Waitlist entries ignored mandate constraints. | A waitlist offer is a hold, with every hold control. A freed time is offered at most three times, then goes public. A merchant's cancellation frees the time only when the merchant says so. Entries obey mandate constraints. |
| Two-party bookings | A tentative booking could last until the start, record nothing when it ended, and keep the booker's reputation credit. Moving a booking kept participants' acceptance of the old contract. A participant's answer ignored their own mandate's money limits. | A tentative booking must settle before the late window opens, and earns credit only when final. A change asks participants again. Answers are checked against the answering participant's own mandate. |
| Contract terms | Whether outcomes were recorded, and the reconfirmation rule, came from current policy rather than the signed offer. Reconfirmation deadlines ignored reschedules and merchant approvals. | Both are terms of the offer. Deadlines move with the booking and start at approval, and a booking whose start has passed is never released. |
| Money | Two confirms in flight could both include the same recorded amount. An anonymous hold escaped a principal's record. A superseded approval refunded the live capture. Refunds owed by an older database were stranded. | A recorded amount is reserved by the first confirm that carries it. An anonymous hold claimed by a principal with records is refused. Money goes back only when the booking says so. Old obligations are adopted into the journal. |
| Wire and tools | Replays skipped service and price limits. The SDK accepted a genuine receipt for a different booking. A moved recurrence from beyond the window was lost. The to-do checker passed a failing suite. Consent text promised records the merchant never makes. | Replays are checked against the result they return. The SDK checks the receipt names the contract it accepted. Overrides are read from both sides of the window. The checker reports the suite's status. Consent text renders only what the offer will do. |

Four lessons.

1. **A new path to a slot inherits none of the old controls.** Waitlist offers and tentative bookings were holds by another name. Both began without the caps and pools that §7 took three passes to build. Every way to occupy a slot now goes through the same checks.

2. **"Every term" has to mean every term.** Two rules lived outside the signed offer. As a result, a policy change could reach existing bookings, and a booker could acquire an obligation they had never approved.
3. **Giving money back needs a fence too.** We had fenced the attempt that commits a booking, but not the one that compensates for losing. Money now goes back according to the booking's state, not because one update failed.
4. **Review is sampling.** Two independent reviews of the same code shared four findings out of 22. A passing suite and one clean review say little about what a third reviewer would find. The case for wire-level vectors and an independent implementation (§11) only gets stronger.

## 10.17 Seventh pass: closing the gaps

The ten seats of §10.13 left 16 scenarios marked todo, each failing on the gap it named. All 16 now pass. Each was rewritten to assert what its seat asked for, not merely that something exists. The sample merchants gained the matching policies: Fades on Queen releases a day 30 days ahead at 10:00, keeps a chair for walk-ins on Saturdays and asked for a deposit after a no-show (removed in v0.4), and Harbourview Dental refuses free-text notes from agents.

| SEAT | THE GAP | WHAT CLOSES IT |
|---|---|---|
| Small merchant | Agents could fill every chair. A deposit was all or nothing per service. | A walk-in reserve that counts only agent bookings. A deposit only for customers with a recorded no-show. |
| Platform | An agent got a fresh rate allowance and a clean reputation at each hosted merchant. | One rate allowance and one reputation per agent across the operator's tenants. |
| Healthcare | Free-text notes were stored as sent. Anyone could book a recall visit. | A clinic can refuse notes before anything is stored, with a link to its own intake form. Services can be for new or for returning patients only. |
| Assistant vendor | A mandate named one merchant. Every charge needed a second prompt. | A mandate can name several merchants, with a shared booking limit the key app keeps. A no-show ceiling the human approves in advance. |
| Multi-location | No shared capacity. A sick technician meant a silent double booking. No brand, no way to extend a job. | Resources with capacity. A resource taken out moves its bookings to a spare, or offers the nearest time. Brands verified in both directions. Extensions as change offers with a new total. |

| SEAT | THE GAP | WHAT CLOSES IT |
|---|---|---|
| Home services | No service address or area. One flat price. | A service address checked against the merchant's area, carried into the offer and the calendar event. Same-day, weekend and after-hours surcharges inside the total. |
| Scarce slots | A rolling window opened evening slots hours after the 10:00 drop. No party size. | Release schedules that open a whole day at once. Parties that consume capacity, tables that combine, prices per person. |

Four limits from earlier sections were lifted on the way. Services with a deposit have a waitlist: the offered hold carries the deposit, and the customer pays it with the confirm. Two-party bookings work at merchants that approve bookings themselves. A participant reads the booking without the booker's records and refunds. A calendar merchant offers nothing it has not read, which answers §11's question about unknown availability. Each rule was then broken on purpose, 27 times in all, and every break made a test fail.

Four pieces were built alongside, in parallel, each by a separate agent in its own copy of the repository:

- **Crash tests.** A merchant process is killed with SIGKILL at the moments around each payment call and restarted on the same files, against a file-backed payment provider that outlives it.
- **Conformance.** 164 wire-level vectors (signatures, mandates, offers, money, durations) and a black-box runner that books, changes, cancels and cleans up against any endpoint, reporting each check against its spec section.
- **An MCP server.** Existing assistants can book through SchCP as tool calls, with the key app's consent text rendered from the signed offer.
- **An AP2 profile.** SD-JWT VC mandates with ES256 keys and selective disclosure, accepted beside the JWS format (§8.6).

**What the crash tests found.** A release voids an authorization the merchant no longer needs, and refunds it if it was captured. It read the authorization, saw "authorized", voided it, and marked the release done. But a capture sent by a merchant process that had just died could reach the rail between that read and the void. A void does not undo a capture, so the deposit stayed captured, with no booking and nothing left in the journal to retry. Three reviews had read this code. Each reasoned about one process and its own calls; the defect needs two processes whose calls interleave at the rail. The release then read the authorization again after voiding, and refunded what it found captured. v0.4 removed the code this defect lived in, with the rest of the money (§10.18).

Three lessons.

1. **Most gaps were vocabulary.** A release schedule, an eligibility rule, a service area or a reserve is one check on the path every hold already takes, because §10.1 put every allocation in one table. Capacity was the exception: it changed what "free" means inside the allocator, and it took the longest.

2. **Closing a gap moves its edge.** An audience list alone would have let one consent book once at every merchant it names, so it needed a booking limit kept somewhere the merchants share: the key app. A walk-in reserve had to know which bookings came from agents. A deposit after a no-show had to read the booking's status, because a merchant that charges nothing for a no-show records no amount.
3. **Kill the process, not the function.** Tests that make a payment call fail exercise the code's error handling. Tests that kill the process at that call exercise what the code left behind. Only the second found the stranded capture.

## 10.18 Eighth pass: taking the money out

The eighth pass started from a question the author asked after reading the reviews: what if nothing were ever charged through SchCP, and it existed only to arrange meetings? Two observations made the answer easy. Most of the hardest work in the first seven passes had been about money, not time. And most meetings have no price at all. Protocol v0.4 takes the money out, and widens who can be a host.

**What went.**

- The payment-provider interface and both providers, in memory and file-backed (332 lines).
- Deposits, the deposit after a no-show, forfeits, `max_deposit`, and the `payment` field on holds and confirms.
- The payment journal, refund state, and the confirm and approval leases, which existed only so a hold could not lapse while money was being captured.
- Adjustments: recorded amounts carried to the next offer for the human to sign. Carrying a debt from one booking to the next is collecting money with extra steps.
- Three error codes (`DEPOSIT_REQUIRED`, `PAYMENT_DECLINED`, `PAYMENT_UNKNOWN`) and the `adjustment_requires_approval` reason.
- Four test files with 31 tests, among them all ten crash tests: they tested a failure class that no longer exists. Tests elsewhere that used a deposit only as a vehicle for something else were rewritten without one.

The engine went from 3,005 lines to 2,628 while gaining open-ended bookings, places and contact lists, and `src/` as a whole from about 9,100 lines to about 8,600.

**What got simpler.** Confirm is one database transaction again. Nothing outside the database happens between checking the offer and writing the booking, so there is no lease to fence, no unknown outcome to reconcile and no refund to journal. Host approval is one transaction too. Of the 65 defects found by the reviews and the crash tests (§10.10 to §10.17), eleven by our count lived in code that is now gone: the payer binding and the swallowed refunds (§10.13), the four in money and leases (§10.14), the four in the money row of §10.16, and the stranded capture (§10.17).

**What was lost, and who answers it now.** A deposit was the strongest protection a scarce slot had, and the payments-risk seat's verdict on v0.3 was to underwrite deposits at confirm. v0.4's answer is that a scheduling protocol is the wrong place to underwrite anything. A business that wants a deposit takes it in its own checkout, and SchCP does not vouch for it. A no-show was still recorded, under terms the human approved, but nothing carried it into a later booking; §10.20 later took the record out too. Whether businesses accept that is now the first question for a pilot (§11).

**What came in.** Several things v0.3's money model had kept out:

- *Unpriced bookings.* A price is optional, and an offer without one has nothing about money in it.
- *Open-ended bookings*, with time set aside and an end the host records (§5.5).
- *Places*: the guest's address, or a park or café the guest names.
- *People as hosts*, with calendars open to their contacts only.
- *Arranging a meeting between two people*: `arrangeMeeting` in the SDK, built at first entirely from existing operations. The ninth pass replaced it with meetings of several people (§10.19).

"Merchant" became "host" and "customer" became "guest" everywhere, from the database to the wire. The suite now has 267 tests: 266 pass, and the one skipped test is still arrival windows.

Three lessons.

1. **The hardest code was in the part that didn't need to be there.** Seven passes made the payment saga correct: leases, fencing tokens, a journal, reconciliation of unknown outcomes, crash tests that kill the process at the rail. Each of those was right, and none of it was about arranging time. The cheapest way to fix a failure class is to stop having it.
2. **A price is a term, not a transaction.** Keeping the stated price and the all-in total in the signed offer kept what the regulator seat asked for: the human sees what they will pay, and a mandate can cap it. It did so without making the protocol a payment rail. Recording outcomes rather than charging them, v0.3's idea, turned out to be most of the way there already.
3. **Generalizing the noun found the missing cases.** Renaming merchant to host was mechanical. Asking "what if the host is a person?" was not. It produced the contact list, the named place, the open end and the meeting with no price, each a short check on paths every hold already takes. We broke each of the new rules on purpose, nine times in all, and every break failed a test.

## 10.19 Ninth pass: several people, and who says yes

The ninth pass started from the author's notes on v0.10's open questions:

- leave recurring meetings open;
- let the guest name the place, but let the other side counter;

- work out meetings of more than two people, where "the contract is three members, and the hashing matches across all three";
- let a host's agent approve by rule, but strongly prefer that a person confirms.

Two more items came off the list as out of scope. Travel time is the humans' business: SchCP arranges the booking, and getting there is up to the people in it. Per-staff skills are the host's own matching. A guest only ever sees what is available, and asking for a particular person is already a selectable resource.

**What was built.**

- *Meeting contracts* signed by the organizer, naming every member, each member's calendar and the hash of that calendar's signed offer. Each calendar checks its own entry when the booking is made (§5.6).
- *Acceptances*, signed yeses to a contract's hash, by the person or by their agent under a rule. Also a *commit* that confirms a meeting at a calendar only when every other member's acceptance of the same hash is presented.
- *Rules* a person signs for their calendar's agent, and a record on every booking of whether the host, a person or a rule said yes. A guest can ask for the person.
- *Counters* from a host waiting to answer, with another place or start. In a meeting, *revisions* of the contract that every member answers again.
- In the sample world, a third person, Priya, and signed rules for Noor and Priya. Noor's calendar moved from instant confirmation to a rule. An unsigned instant yes from a person's calendar was exactly what the author asked us to avoid.

The engine grew by about 440 lines, and a protocol module of about 180 lines verifies contracts, acceptances and rules. Nine tests cover meetings and answers. The two-family playdate now goes through a contract too, and the sample site gained a three-family scenario.

**Decisions.**

- **Every calendar checks only its own entry.** No host needs another's keys or talks to another host. The contract holds together because each member's calendar checks its own offer against the same hashed document. What a calendar cannot check is who the other members are: that is the organizer's word, and a member's agent should show them by its own contact names.
- **Rules moved onto the wire.** v0.10 said rules belonged in the host's own agent. A meeting changed that. Ellis's calendar has to accept Priya's yes without having seen it given, so a yes by rule carries the rule, signed by Priya and naming the key that used it.
- **The organizer's agent coordinates.** Holds are the first phase, every member's acceptance is a vote, and the commits are the second phase. No coordinator service is needed. If a commit fails partway, the agent cancels the others. A person may take hours to answer, so the short arranging mandate is followed by a narrow one that covers only these bookings, until the start.
- **A confirmed meeting is not revised.** Moving it means arranging it again. That is the smallest rule that keeps "everyone agreed to the same hash" true. v0.14 replaced it with a revision that waits beside the

confirmed meeting until everyone agrees (§10.22).

- **Businesses keep instant booking.** Their published terms are their standing yes, and the guest's person always approves. A person's calendar in a meeting always needs an acceptance.

Writing the spec from the code found four places where the code was looser than the words:

- a guest's request for the host's person was forgotten on an ordinary booking, so a later yes by rule got through;
- an acceptance could be dated back to a time when an expired rule was still valid;
- the new claims were checked by hand rather than against their schemas;
- a revision's deadline ignored how long the host promises to take to answer.

All four were fixed, each with a test; an acceptance must now be dated between the moment its terms existed and now. The conformance package gained 27 vectors for contracts and acceptances, and the runner gained two live checks that need a contact's key to run.

We broke fifteen of the new checks on purpose. Fourteen failed a test at once. The fifteenth, the rule that a revision's number must go up, did not, because no test tried to roll a calendar back to an earlier contract. One does now.

Two lessons.

1. **A shared hash is only shared if everyone checks their own part.** Nobody verifies the whole contract. Each calendar verifies the one line about itself, and the hash makes those lines one document.
2. **The question is not whether a person or a machine said yes, but whether anyone else can check which.** An agent's yes under a rule its person signed is checkable; an agent's yes on its own is not. Making the rule a signature let us allow rules and still say plainly when a person has not seen something.

## 10.20 Tenth pass: late cancellations and no-shows are the host's business

Reading a condensed review of v0.11, the author stopped at outcome records: "it makes no sense for us to have no shows and cancellations recorded, let's leave that business logic up to the hosts agent/system (since we are not dealing with money any longer)."

The record was the last piece of money left in SchCP. v0.3 recorded late outcomes instead of charging them. v0.4 stopped charging anything, but kept the record, with the fee the offer stated as information, so a host could bill it outside the protocol. With no money in SchCP, a fee the protocol states but never collects is the host's policy written into the protocol, and the record of it is the host's ledger kept on the host's behalf.

**What was removed.**

- Cancellation tiers, no-show fees and the late-change rule, from discovery and from every offer, so the consent text no longer mentions them.

- Outcome records on bookings and receipts, and the table behind them.
- The cancel quote, which existed to say what cancelling would record.
- The 60-second undo, whose only meaning was "nothing recorded". Every cancellation is now free and records nothing.
- `max_no_show_fee` in mandates and the closed-mandate policy `if_no_show_fee`. A host that wants the human's own signature asks for it on every offer; Harbourview Dental does.
- The rule that a booking with participants must settle before the first late tier. It now settles by its start at the latest.

**What was kept, and why.** A host may still mark a past booking attended or a no-show, as a status the guest's agent sees, with no amount. That status is what the abuse controls stand on. A principal becomes a returning guest only by keeping an appointment (§7.2), so throwaway identities flooding a host stay capped, and a no-show earns none of that standing. We asked whether to drop the marking too. That would have been simpler, but booking and not turning up would then earn standing, and the one bound that works without money would weaken.

**What it cost.** A host can no longer show its cancellation and no-show terms through the agent's consent text. Terms the customer had verifiably seen were one of the things §9.4 offered businesses. A host that bills a no-show now does it on terms the guest agreed to elsewhere. The suite lost two tests: the undo, and a rule about whether outcomes are recorded. About thirty others were rewritten to check that cancelling, moving and missing a booking record nothing and state no amount. The conformance runner's two cancellation checks became one, `cancel.free`. The vectors lost the fee fields from their example offers. About 380 lines of source went. The same pass dropped the database migration and its test: nothing has been released, so there is nothing to migrate, and the protocol makes no compatibility promise between drafts.

That last point closed two gaps. Signed documents (offers, mandates, meeting contracts, acceptances and rules) had been left open to fields they don't define, the usual choice when older verifiers must accept newer documents. With no such promise to keep, they are closed like the requests: the agent SDK refuses an offer that adds a term SchCP doesn't define, even one the host signed, before the human sees it, and a new vector shows one with a cancellation fee. The consent text now also states a reconfirmation the offer binds the booking to: when it is due, and what silence does. Two tests and one vector were added.

One lesson.

1. **Taking money out is not finished until its bookkeeping is out too.** A record of a fee nobody collects looked like a neutral fact. It was a host's policy carried by the protocol, with a schema, a quote endpoint, an undo window and a consent clause to support it. Once the record went, each of those went with it.

## 10.21 Eleventh pass: tested with AI assistants on both sides

Every earlier pass tested SchCP with code: the reference SDK on one side, the reference endpoint on the other, and tests written by people who knew what each field meant. The MCP server (§10.17) was built so that existing assistants could book without a new client, but no AI assistant had used it. The eleventh pass handed both sides of each booking to AI assistants, and fixed whatever they misread or could not do until they stopped finding anything.

**The harness.** Each scenario ran a local gateway around the reference engine and the simulated hosts of §10.0. The gateway wrapped the MCP tools (the same code as the MCP server) and each host's side, exposed both over plain HTTP, and logged every call. AI assistants (Claude Haiku) played the people:

- *Guests:* Noor, arranging a playdate or a coffee with Ellis and Priya; Sam, Kim and Alex, booking, moving and cancelling at Rosie's Nails, Maple Street Auto, Harbourview Dental and Fades on Queen. Each held one person's key app through the tools and reported to its user at the end.
- *The host side:* the front desk of Rosie's Nails, which approves each booking, and the agents keeping Ellis's and Priya's calendars. They read what arrived, in words, and accepted, declined, countered or backed out. Maple Street Auto confirms instantly, so nobody had to answer for it.

Assistants waiting for an answer polled the gateway rather than taking webhooks. A round ran three scenarios side by side, such as a pedicure moved at a salon that approves each booking, a playdate or coffee arranged across three calendars, and a haircut with a particular barber, with five to seven assistants across them.

**The loop, and when to stop.** After each round we read every assistant's report to its user and the gateway's log of what it actually did and was told. Defects were fixed in the engine, the spec and the tools. Wording an assistant misread was rewritten in the tool descriptions, the results and the consent text. Each fix went in with its tests, and the next round ran. The rule was to stop after three clean rounds in a row. A round was clean when no assistant misbehaved, gave its user a wrong report, or hit a defect. A cheap wording fix noted in a round did not stop it counting as clean, and a problem in the test harness alone did not count. It took 25 rounds. Rounds 17, 19 and 21 were clean, but each was followed by one that was not; 23, 24 and 25 were clean in a row.

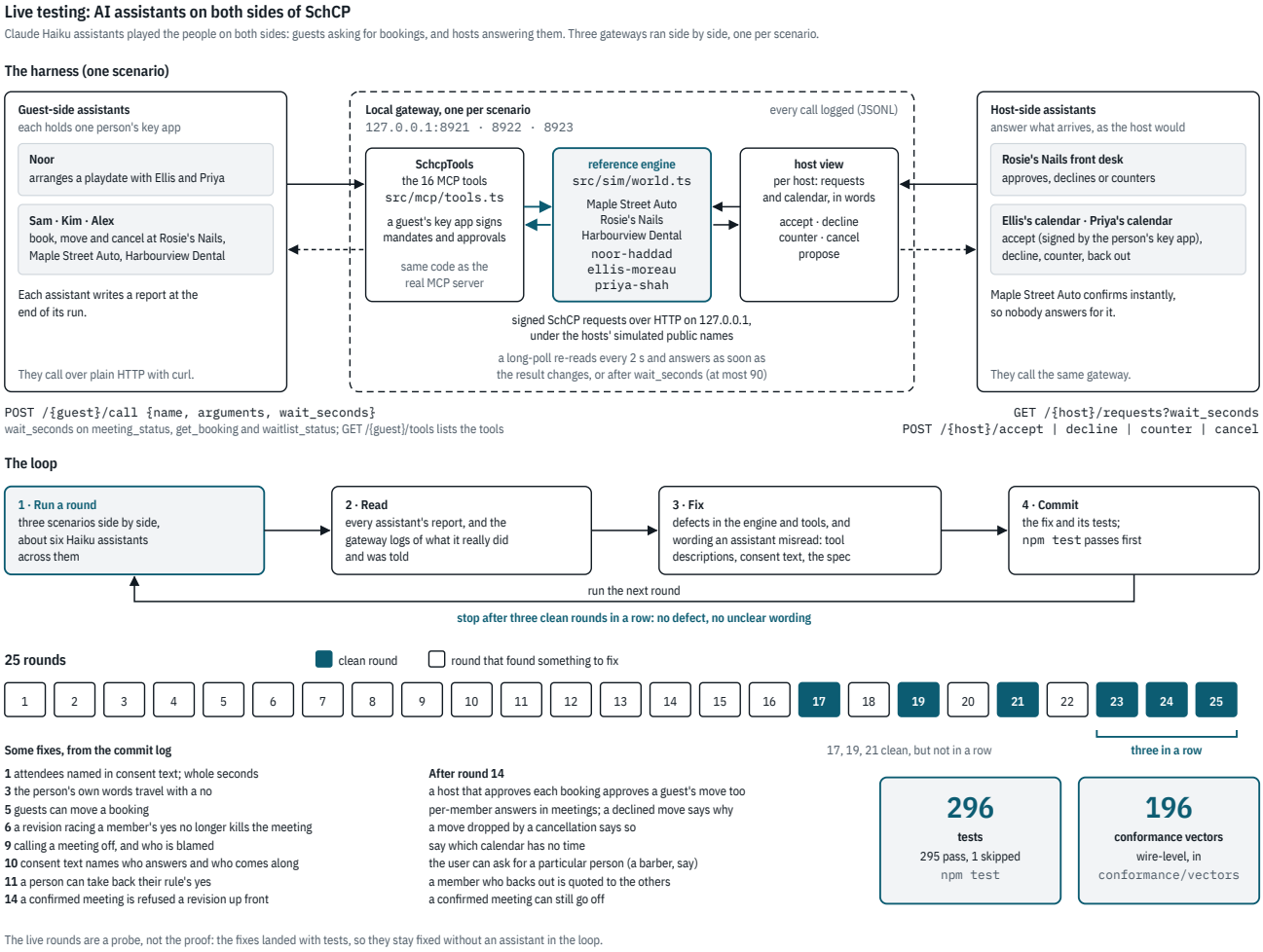



**Figure 18.** How the live testing was run: the gateway for one scenario, with assistants on the guest side and on the host side calling it over HTTP; the loop after each round; and the 25 rounds, with the clean ones filled. The counts are those at the end of live testing. The fixes landed with tests, so they stay fixed without an assistant in the loop.

**What the assistants got wrong, and why it mattered.** Few of the failures were carelessness. Most were an assistant reading exactly what it was given and drawing a reasonable, wrong conclusion:

- `approve_and_book` returned `booked: false` for a booking waiting for the host's answer, which read as a failure (round 2).
- `discover_host` called a policy `human_signature`, which assistants read as "the host's yes is not signed" (round 5).
- Prices said they were "before any tax or fees", and an assistant told its user that HST might be added to an all-in total that had none (round 13).
- An organizer's assistant took the arranging hold's ten-minute expiry for the members' deadline to answer (round 10). Another read "11:58 p.m. today" as already past (round 19).
- A member's assistant could not tell who "Theo" was in the consent text, and held off saying yes (round 10).

- From round 15 to round 21, assistants kept reading `approved_by: "agent"` as "the user did not approve", although the user had approved and the field says only what the user's key app signed.

Other failures were things an assistant could not do or could not see. Asked to move a booking, one had no tool for it, and cancelled and rebooked instead (round 5). A time freed from a waitlist was held for a user, and no tool could see it (round 4). One cancelled a meeting booking by booking, and another calendar briefly still showed the meeting (round 9). One invented a surname in a note to a host, which cannot be taken back (round 13).

When an assistant reads a protocol, its wording is part of the protocol. A field name, a tool description or a refusal is what the assistant reasons from, and then repeats to a person. Each misreading above would have reached someone as a wrong report: a booking said to have failed, a tax that was not there, a consent that seemed not to be theirs. The signed bytes were right each time. What they were called was not.

**What was fixed.** 28 commits, most of them naming the round that found them. By kind:

- *The protocol.* A host that approves each booking approves a guest's move too (round 15; §5.7). Sam had moved a confirmed pedicure at Rosie's Nails, and it went straight to the new time without the salon's yes. A declined move is kept on the booking as `declined_change`, for agents that poll (round 16). A meeting member may carry its own `answers` (round 16). Offers carry `resource_name` inside the signed claims, so the name the person approves is the host's own (round 20). A member may take back a yes until the meeting is final (round 11), and one who backs out is quoted to the others (round 22). A confirmed meeting is refused a revision before anything is held or asked (round 14), so it is changed only by calling it off and arranging it again. A waitlist window too short for the appointment is refused (`window_too_short`, round 5). A business's own answer is recorded as `by: "host"`, not `"person"`, which assistants had read as a private individual rather than the salon (rounds 15–25; spec §7.9).
- *The tool interface.* Four new tools, for 16 in all: `change_booking` (round 5), `waitlist_status` and `leave_waitlist` (round 4), and `cancel_meeting` (round 9). `discover_host` lists `choose_from`, and `find_slots` and `hold_slot` take `with`, so a user can ask for a particular barber (round 20). `find_slots` takes `daily` hours, so a page of daytime slots cannot crowd out the evenings someone asked for (round 22). `discover_host` lists the host's taxes, so a total above the list price is explained before a hold (round 19).
- *Wording.* `approve_and_book` says `user_signed`: "this exact offer", or "only the limits: service, time window, price ceiling" (rounds 21 and 24). Outcomes say `requested` or `confirmed` rather than `booked: false` (round 2). Deadlines come with relative times, "about 10 minutes from now" (round 19). Consent text names who answers how and who comes along, with their relationship: "with Maya (Noor's daughter) and Theo (Ellis's son) coming along" (round 10). Prices say "list price" (round 13).

| ROUNDS | WHAT CAME UP | SOME FIXES (ROUND) |
|---|---|---|
| 1–4 | Results read the wrong way; things no tool could see | Attendees named in consent text (1); `outcome: "requested"` instead of `booked: false` (2); a no carries the person's own words (3); `waitlist_status` and `leave_waitlist` (4) |
| 5–8 | Things no tool could do; dead ends nobody reported | `change_booking` and `window_too_short` (5); a revision racing a member's yes no longer calls the meeting off (6); a refusal over the price limit says what to do next (8) |
| 9–14 | Meetings: going off, who is blamed, who answers | `cancel_meeting`, and a no blamed before any cancellation (9); `answer_by` in `meeting_status` (10); taking back a rule's yes (11); a decline is the person's own answer (12); a confirmed meeting refused a revision up front (14) |
| 15–16 | Moves at a host that approves each booking; answers per member | `awaiting_host`, `booking.change_approved` and `booking.change_declined` (15); `declined_change` and per-member `answers` (16) |
| 17–25 | Mostly wording, in and around the clean rounds | A service found by a person's words (17); which calendar has no time, taxes, relative expiries (19); `choose_from`, `with` and `resource_name` (20); `user_signed` (21, 24); backing out quoted, `daily` hours (22); `"24:00"` as the end of a day (23) |

**What did not change.** Money stayed out entirely, and so did records of late cancellations and no-shows. None of the fixes needed them. Holds, mandates, signed offers accepted by their hash and one contract hash per meeting stood as they were; what changed was who may say yes to what, and what each side is told.

**What the clean rounds showed.** In each of the last three rounds, every assistant did what its person asked or said why it could not, and told its person correctly what had happened, as a guest and as a host. That shows the tools and their words are usable by one model family in the scenarios we wrote. The fixes themselves are not a matter of which model reads them: each landed with tests, the suite grew from 282 to 296, and they stay fixed without an assistant in the loop. The conformance vectors grew by 23, to 218. One is for an offer that names the person a guest chose (`resource_name`). The other 22 (`moves.json`) are for a guest's move at a host that approves each booking, as the guest's agent reads it from signed receipts and from `booking.change_approved` and `booking.change_declined` webhooks: waiting for the host, made, declined, lapsed, or dropped with the booking, and the inconsistencies an agent must reject, such as a move marked as waiting whose booking already shows the new time, or a receipt whose booking carries a charge. The JSON Schema gained `Booking`, `Proposal` and `ReceiptClaims`, closed like offers, and an agent refuses a receipt that fails them.

Three lessons.

1. **When an assistant reads the protocol, its wording is part of the protocol.** `approved_by: "agent"` was precise on the wire, and stayed there until v0.14 replaced it with `consent.approved` (§10.22). Read by an assistant without the spec, it said the opposite of what happened. The tool now states the same fact from the user's side. Names, descriptions and refusals need the same review as the fields they describe.
2. **Assistants try the paths nobody wrote a test for.** A guest moving a booking at a host that approves each booking had no test because nobody had thought to try it. Nor did a person taking back the yes their rule had given. An assistant asked to do what a person wants does things in the order a person would.
3. **Put the reason where the reader looks.** Several fixes moved an answer from a webhook or a later refusal to where an assistant looks first: `declined_change` on the booking, the notes policy in `discover_host`, the refusal of a confirmed meeting's revision before anything is held, the members' deadline in `meeting_status`.

**Limits.** The hosts were the simulated ones of §10.0, and the people were played by models: no real business, no real family, no real calendar. One model family (Claude Haiku) played every part on both sides, so the assistants may have shared misreadings and blind spots that another model would not, and another model may read the tools differently. We wrote the scenarios, read the reports and decided what counted as clean. Three clean rounds in a row show that the tools are usable in those scenarios, not that every assistant will read them the same way.

## 10.22 Twelfth pass: deciding the open questions

By v0.13 the paper carried more than twenty open questions, most of them since the ninth pass. On 17 October 2026 the author answered fourteen of them, each put as a plain choice with an everyday example and a recommendation, and this pass built what the answers asked for. The suite grew from 297 to 321 tests, and the conformance vectors from 218 to 232.

| QUESTION | DECISION | WHAT CHANGED |
|---|---|---|
| Repeating meetings ("every second Saturday") | Not in v0.4 | Each occurrence is its own meeting; an assistant arranges the next one |
| Changing a meeting everyone already agreed to | Keep the old time until everyone agrees | A revision waits beside the confirmed meeting; a no, or no answer before it starts, leaves it as it was (§5.6) |
| A café inside a meeting | Allowed; its offer is its yes | Members with `business: true`; Bellwoods Café in the sample world |
| Who the other members are | Show them by your own contacts | Otherwise "someone you don't know", with the organizer's name for them in quotes (a SHOULD) |

| QUESTION | DECISION | WHAT CHANGED |
|---|---|---|
| Finding a place both sides like | Out of scope | The guest names it and the other side may counter, as before |
| Saying "I take part in meetings" | A discovery flag | The `meetings` capability, checked before anything is held |
| Accessibility needs as structured requests | Not yet | Still open (§11) |
| How long a host keeps booking records | The host states it | `retention` in discovery, in every offer and receipt, and in the consent text |
| Receipts as evidence | Say how the person agreed, under which mandate, and the retention | `consent: {approved, mandate, mode}` and `retention` on every receipt |
| Standing from attendance the host never marked | A host's silence still counts as attended | Unchanged; most small hosts will not mark every visit |
| A shared booking limit after a cancel | The booking is given back | Released on a cancel, a decline or a lapse, and retried while the key app cannot be reached |
| A guest's track record across hosts | Left out | It would let hosts link a person across places; each host keeps its own view |
| Approving without reading | Short guidance | SHOULDs for assistants (§8.2); `differs_from_request` in the MCP server |
| Join the business-booking efforts, or stand alone | Stand alone | SchCP is its own protocol for scheduling people (§4, §11, §12) |

One answer changed words rather than behaviour. Asked whether cancelling should give a booking back to the "budget" kept in a person's "wallet", the author pointed out that both words sound like money. Neither held any: the wallet was the app that holds the person's signing key, and the budget a count of bookings. They are now the *key app* and a *shared booking limit*, in the spec, the code, this paper and the explainer video.

The same pass closed smaller gaps. Holds, waitlist entries and booking responses gained closed schemas, so an agent now refuses any of them carrying a field the spec does not define. The probes of the v0.2 and v0.3 reviews were retired: each of their 46 findings is a regression test or went away with the money. A script now

fails when a count in these documents disagrees with the code. And one test, which had timed a capped calendar expansion by the clock, failed on a heavily loaded machine; it now measures CPU time. That is the likely cause of the one unexplained failure in live testing.

Three lessons.

1. **A question put as a plain choice gets answered.** Most of these had stayed open because they were written for a standards reader. Each took a minute once it came with an example, the options and a recommendation.
2. **Words carry the design.** "Wallet" and "budget" were accurate to a specialist and wrong to everyone else. Live testing taught the same lesson about `approved_by` (§10.21).
3. **A good pattern pays twice.** Changing a confirmed meeting reuses the shape of a guest's move at a host that approves each booking: keep what was agreed, hold the change beside it, and make it all or nothing.

## 10.23 Thirteenth pass: a second implementation from the spec alone

Agreement between a serializer and a client written by the same hands is weak evidence (§11). So an agent that could read only the spec, the JSON Schema and, at the end, the conformance vectors, and never the reference code, wrote SchCP's verifiable parts again in Python: request signatures, mandates and their chains, money and durations, offers and holds, receipts, a guest's move as an agent reads it, meeting contracts, acceptances and rules, and a client that books against the sample site. It does not run a host endpoint; the checks a host makes that are pure functions, it implements. It is in the repository under `implementations/python/`.

**First pass.** It passed every vector and booked at three sample hosts (instant, approved by the host, and a clinic that needs the person's own signature). But about 75 of its 238 cases passed only because the vectors or the schema gave away a rule the text never stated, and it listed 24 findings. The four that would stop two implementations working together:

- **The exact form of a signed request.** RFC 9421 lets the signer choose the label and the order of what it covers. The reference refused anything but its own choice, and only the vectors showed it.
- **How a sub-agent's mandate is built and checked.** The key that signed it, which principal it speaks for and which checks apply to each link were in the schema and the code, not the prose.
- **Money arithmetic.** Decimal places per currency, rounding a tax, and never adding across currencies were in the vectors only.
- **Validating dates.** The spec relied on the schema to reject an unparseable date, but JSON Schema treats `format` as optional, and Python's validator skips it.

Each finding is now settled in the spec, the schema or the sample site (the spec's §2.1, §3.2, §3.3 and §5.7 grew most). Several changed the reference too. A cancellation fee tucked inside an offer's `slot` used to pass the "closed" schema, which was closed only at the top; every object inside signed claims is now closed. The reference answered a replayed request with the generic reason, not the registered one. Its client tried every

host key instead of the one the signature names, accepted JSON with a repeated member, did not check that an offer's lines add up to its total, and computed percentages with a binary float. One vector had allowed two outcomes, refused at verification or at the action; the spec now picks one. And nothing said how a client outside the sample site's process could reach its simulated hosts, which listen on ports the system picks; the site now publishes the map.

**Second pass.** A fresh agent, with the same restrictions and no sight of the first implementation, wrote it again from the revised spec, and opened the vectors only when it was done. Its first run passed 218 of the 233. Eight of the 15 failures needed a rule found only in the vectors: how to write a duration, which the spec now states. One was a duty the spec gave the host but not the agent. The rest were its own bugs. It then booked and cancelled at the sample site on its first attempt. Its probes of the reference found four places where a careful reading and the reference differed: an expired mandate reported as merely invalid, an agent key accepted with its private part attached, and two orders of checks that decide which reason is reported. The first two are fixed; for the order, the spec now says a host may report any one of several failures and an agent must not rely on which.

What this does not show. Both implementations were written by AI agents of the model family that wrote the reference, and an implementer who shares a writer's habits may share its blind spots. Neither runs a host endpoint. And the vectors do not yet cover shared booking limits, the closed-approval path, committing a meeting, revocation or the SD-JWT profile, so agreement there is untested.

Three lessons.

1. **Vectors leak the spec.** Passing every vector felt like proof the text was enough. It was not: a quarter of the cases were passed by reading the answer key. The useful measure is the first run, before the implementer has seen the vectors.
2. **Reasons and order are part of the interface.** No vector disagreed about whether a request is refused; several readings disagreed about why. An agent that acts on the reason ("ask for a fresh mandate") needs the reason to be the same everywhere.
3. **"Closed" has to go all the way down.** The rule that a term the protocol does not define never reaches the person held at the top of an offer and failed one level in.

# 11 Open questions

- **Governance.** SchCP stands on its own (§10.22), so its governance is its own question. AdCP's lesson is that a neutral body is necessary but not sufficient: even with one, a founding vendor's weight draws scrutiny. A protocol owned by one author is just an API. Before any version is Final, SchCP needs implementers and owners beyond the people who wrote it.
- **Living beside the business-booking efforts.** UCP's Local Services charter and Wix's USP cover booking a business with stronger backing, and SchCP keeps its envelope close to theirs. Open: whether a bridge to them is worth building, so that one endpoint can speak both, and whether businesses will run two protocols at all.

- **Will businesses accept a protocol that never charges?** v0.4 takes no money at all, at booking or after it. A business that wants a deposit or a no-show fee takes it outside SchCP, through its own checkout. For a salon whose no-show fees are its main protection, that may be a reason not to adopt. SchCP no longer records late cancellations or no-shows for the host either: that is the host's own system's job. Pilots should measure whether the controls of §7.4 hold no-show rates where deposits did, and whether hosts miss a shared record of outcomes enough to want one back.
- **Who hosts people?** Personal-calendar services are the natural operators (§9.3). Whether they want that role, and how a person moves their calendar between them without losing their contacts or their standing, is untested.
- **Which attesters, at what privacy cost?** v0.2 gives hosts a way to trust some newcomers without a central authority (§7.5). Which checks deserve that trust (a phone, a card, an ID) is an empirical question only real traffic can answer. An attestation is also a cross-host identifier, and an unlinkable version would need per-host keys and selective disclosure. AdCP's warning still applies: "every centralized trust authority eventually becomes a tax."
- **Tuning.** The seven-day half-life, the five-outcome floor and the TTL tiers are guesses that behave sensibly in tests. Real abuse data should set them.
- **Scope above one host.** v0.9 built a mandate for several hosts with a shared booking limit the key app keeps, brands verified both ways, and operator-wide rate limits and reputation (§10.17); a cancelled booking now gives its unit back. Open: whether key apps will run that counting service at all, which makes the key app's availability part of booking. Searching across a brand's locations is out of scope: a brand claim only makes the relationship checkable (§2.4).
- **Accessibility needs.** Refusals say why, and every one names a way to book without an agent. Still open, and deliberately not yet built: needs such as step-free access, a quiet space, extra time or a support person as structured requests that shape allocation, appear in the signed offer and can never raise the price.
- **Regulated data.** A clinic can refuse free-text notes and point to its own intake form, participants say when the attendee is not the signer (a parent, a substitute decision-maker), and since v0.14 every host states how long it keeps booking records. Still open: SchCP does not check that a host keeps its word, and a booking at a clinic is itself health information that the protocol says nothing more about.
- **Platform risk.** PAP is building the identity and session layer for personal agents. Its `operations` extension defines `approved_by` as `user` or `standing_permission`, and says both work on trust.[39] A booking mandate could ride on it through a proposed extension (a third value backed by a principal-signed closed approval the company can verify), or be made unnecessary by it. Engaging early is better than being surprised.
- **A host written by someone else.** Two implementations from the spec alone now agree with this one on every vector (§10.23), but both are of the agent's side and the host's pure checks, and both were written by AI agents from the same model family as the reference. Still untested: a host endpoint built elsewhere, by people, and the areas no vector covers yet (shared booking limits, the closed-approval path, committing a meeting, revocation, the SD-JWT profile).
- **Live testing was narrow.** The assistants of §10.21 worked against simulated hosts, played people in scenarios we wrote, and all came from one model family, Claude Haiku. Other models, a real business's

staff and real people's assistants may misread what these read correctly, or the reverse. A run with other model families, and with a real host answering, is the next test of the wording.

**Decided, and no longer open** (§10.22). Recurring meetings are not in v0.4. Changing a confirmed meeting, businesses as meeting members and showing members by your own contacts are built. Finding a place both sides like is out of scope, as are class passes, memberships, lotteries for scarce releases, arrival windows, matching staff to services and travel time. Records retention and receipts as evidence are built, and consent fatigue has short guidance in the spec. A cross-host reputation is left out, for privacy. Standing is still earned by a booking whose time passed without being cancelled or marked a no-show, so a host's silence counts as attendance. And SchCP stands on its own rather than as a profile of the business-booking efforts.

# 12 Conclusion and ask

Agents can already talk, browse and pay for people. The envelope for booking a business's time is converging in 2026 under UCP, USP and the Local Services charter, and AP2 and ACP carry payment. What none of these efforts defines is how a host, whether a business or another person, can verify what the human allowed an agent to commit them to: the time, the place, any price, the right to change it later, and proof that the human saw the exact terms. This paper offers a worked answer, built and then broken repeatedly: by an adversary, by our own open questions, by ten business seats, by three deep reviews, by crash tests, and by AI assistants using it on both sides of each booking.

Its v0.4 makes one decision the rest of the paper argues for: SchCP never moves money. That makes it smaller, it removed a whole class of failures, and it fits the meetings most people actually have, most of which have no price. v0.14 adds a second: SchCP stands on its own as a protocol for scheduling people, borrowing from the business-booking efforts where they fit rather than folding into them, because what it defines (what the person allowed, and meetings between people) is what they leave out. It is meant as a first step toward agents that coordinate people's time on their behalf, one signed commitment at a time.

**The ask.** We ask for three things, in order:

1. **Review the spec.** Especially the constraint types ( `services` , `window` , `max_price` against a stated all-in total, `max_bookings` , a shared booking limit, and management scope); the rule that a booking's money terms are disclosures, never charges, and that what a late cancellation or a no-show costs is the host's own business; meetings as one contract every member accepts; and one approval-profile decision: where a host asks for the human's approval of the exact offer, must the closed approval be signed by the human's key over the host-signed offer, rather than closed by the agent?
2. **Implement it independently.** A second implementation, or a second SDK written from the spec alone, run against the wire-level vectors:
   - mandate-chain checks;
   - signed offers and receipts, including how the person agreed and how long records are kept;
   - idempotent replay across principals;

- terms fixed across a policy change;
- contract-hash mismatches and change chains;
- a cancellation that is free and records nothing;
- open-ended bookings and their end;
- contacts-only access;
- meeting contracts, acceptances by person and by rule, commits, changes to a confirmed meeting, and businesses as members.

This is our own next step too.

3. **Pilot it.** With one shop and one personal-calendar provider, and with assistants from more than one model family.

A first vector package exists ( `conformance/` , wire-level vectors and a black-box runner); the next step is to run it against an endpoint we did not write. Contact: Cantal (cantal.ca).

**A path to adoption.** Start with two deliberately narrow cases. For businesses, a limited auto-service catalog: oil changes, tire changeovers, inspections and alignments, whose duration and price are fixed, plus a diagnosis drop-off as an open-ended booking with no price. Repair estimates, and anything that depends on parts, hand back to staff. For people, playdates and coffees between contacts on one personal-calendar service, where nothing is priced, the contact list does the gatekeeping, and nothing is confirmed until every person has said yes. The walkthrough and the playdate (§6) are the first cases. The MCP server that lets existing assistants book without a new client, tested live with assistants on both sides (§10.21), and the black-box runner any endpoint can be checked against, are built (§10.17); what they need now is a real shop, a real calendar provider and a second implementation.

Licensing is adapted from AdCP's model so that the terms are not a reason to hesitate.[40] The code and tests are Apache 2.0, and the specification and this paper are CC-BY-4.0. Contributors grant a royalty-free licence under their Necessary Claims to anyone implementing a Final Specification (IPR_POLICY.md in the repository). AdCP's patent commitment runs through its foundation, whereas SchCP's grants directly to implementers. No SchCP version is yet Final.

# — Sources

Full annotated research reports, with verification labels for every claim, are in `research/` : `adcp.md` , `scheduling-standards.md` , `agent-commerce-delegation.md` and `booking-platforms.md` . Figures from vendors and the press are marked as such in the text. External pages were checked on 13–14 October 2026. Protocol drafts are changing quickly, so every comparison describes the page as it read on those dates.

1. AdCP 3.2 documentation, design principles and known limitations; AgenticAdvertising.org governance; press coverage (AdExchanger, Digiday, Marketing Brew). docs.adcontextprotocol.org; agenticadvertising.org/governance.

2. AdCP, "adagents.json Tech Spec" and "Understanding Authorization." docs.adcontextprotocol.org/dist/docs/3.2.3/governance/property/adagents.md
3. AdCP, "Task Lifecycle." docs.adcontextprotocol.org/dist/docs/3.2.3/building/by-layer/L3/task-lifecycle.md
4. AdCP, "Proposal negotiation." docs.adcontextprotocol.org/dist/docs/3.2.3/media-buy/product-discovery/proposal-negotiation.md
5. RFC 5546, iCalendar Transport-Independent Interoperability Protocol (iTIP). rfc-editor.org/rfc/rfc5546
6. Hold prior art: Google Actions Center `CreateLease`; Cal.com slot reservations; Wix reservation lifecycle; airline GDS ticketing time limits and agency debit memo policies. See research/scheduling-standards.md §8.
7. RFC 6047, iMIP, §2.2.1. rfc-editor.org/rfc/rfc6047
8. draft-desruisseaux-ischedule-05 and IETF datatracker history; IETF appsawg iSchedule wiki. datatracker.ietf.org/doc/draft-desruisseaux-ischedule/
9. RFC 8984 (JSCalendar); draft-ietf-calext-jscalendarbis-22, Appendix A.2.2.2.
10. Google Actions Center booking server (v3) and Appointments Redirect migration guide. developers.google.com/actions-center
11. Universal Commerce Protocol specification; lodging booking capability, PR #780. ucp.dev; github.com/Universal-Commerce-Protocol/ucp/pull/780
12. UCP Local Services Domain Tech Council charter, issue #922, filed 2026-10-09. github.com/Universal-Commerce-Protocol/ucp/issues/922
13. Wix, Universal Scheduling Protocol (draft 2026-08-20). github.com/wix/universal-scheduling-protocol
14. AP2 v0.2 specification and Agent Authorization Framework; Mastercard Verifiable Intent; Visa and Stripe documentation on authorizations and merchant-initiated transactions. ap2-protocol.org; github.com/agent-intent/verifiable-intent
15. Agentic Commerce Protocol: agentic checkout and delegate payment RFCs; Stripe Shared Payment Tokens. github.com/agentic-commerce-protocol/agentic-commerce-protocol
16. Personal Agent Protocol, Draft 0.1 and the `operations` extension; Sierra announcement, 2026-10-06. personalagentprotocol.org; sierra.ai/blog/introducing-personal-agent-protocol
17. Google, "AI Mode" agentic features (2025-08-21) and the beauty and wellness expansion (Nov 2025). blog.google/products/search/ai-mode-agentic-personalized/
18. RFC 9421, HTTP Message Signatures; RFC 9530, Digest Fields; RFC 7638, JWK Thumbprint; RFC 8037, CFRG curves in JOSE.
19. RFC 5545, iCalendar; RFC 4791 and RFC 6638, CalDAV and CalDAV Scheduling; draft-ietf-calext-vpoll-07.
20. IETF Web Bot Auth working group; Cloudflare Web Bot Auth documentation; Visa Trusted Agent Protocol.
21. A2A specification (Agent Card, task states); Model Context Protocol, revision 2026-07-28.
22. RFC 8693 (Token Exchange), RFC 9396 (Rich Authorization Requests), RFC 9449 (DPoP), RFC 9901 (SD-JWT); Biscuit; UCAN.
23. schema.org `ReserveAction`, `Reservation` and `ReservationStatusType`.
24. Square Bookings API; Mindbody Public API pricing; Vagaro API reference; Jane Developer Platform; Phorest API; Calendly and Cal.com APIs; Microsoft Graph Bookings.

25. Calendly MCP server (2026-03); Square MCP server (beta); Cal.com hosted MCP.
26. Supply-side landscape, market data and platform verification, including docs.booksy.com (HTTP 401, checked 2026-10-13), API Evangelist's Booksy profile, U.S. Census nonemployer statistics via secondary compilation, NFIB 2025 technology survey, Dantas et al., *Health Policy* 2018 (no-show review), and Fresha and OpenTable vendor figures. See research/booking-platforms.md.
27. OpenTable 2026 product release and restaurant impact report (vendor figures).
28. CNN Business, "Now AI is trying to gobble up dinner reservations," 2026-09-23.
29. CX Network, Fresha native ChatGPT and Claude integrations, 2026-09-09.
30. Haeder S. and Xu W., *Health Affairs Scholar* 2025;3(4): qxaf077 (US adults' scheduling channels; survey fielded October–November 2024). academic.oup.com/healthaffairsscholar/article/3/4/qxaf077/8108999
31. Booksy Biz pricing, Boost terms (US). biz.booksy.com/pricing
32. Fresha pricing, new-client marketplace fee. www.fresha.com/pricing
33. US Census Bureau, industry profile for NAICS 812112 Beauty salons (employer establishments, 2023). data.census.gov/profile/812112_-_Beauty_salons?codeset=naics~812112&g=010XX00US
34. Dantas L.F. et al., "No-shows in appointment scheduling: a systematic literature review," *Health Policy* 2018. pubmed.ncbi.nlm.nih.gov/29482948/
35. Treatwell, company post on UK salon no-show cost (company estimate; method not published). www.linkedin.com/posts/treatwell-com_treatyoursalonwell-supportsmallbusiness-activity-7297559561459167233-h0rQ
36. Universal Scheduling Protocol, security requirements (hold-abuse controls, booking-scoped credentials). usp-protocol.dev/security/
37. AP2, Agent Authorization Framework (open and closed mandates, constraint extensibility). ap2-protocol.org/ap2/agent_authorization/
38. Google, AP2 v0.2 and donation to the FIDO Alliance, 2026-04-28. blog.google/products-and-platforms/platforms/google-pay/agent-payments-protocol-fido-alliance/
39. Personal Agent Protocol, Draft 0.1, `operations` extension. personalagentprotocol.org/docs/extensions/operations
40. AdCP, Intellectual Property Rights policy. github.com/adcontextprotocol/adcp/blob/main/IPR_POLICY.md
41. US Census Bureau, Nonemployer Statistics 2023, US file, NAICS 812112, all legal forms, all receipt sizes: 838,264 establishments (retrieved 2026-10-14). www2.census.gov/programs-surveys/nonemployer-statistics/datasets/2023/historical-datasets/nonemp23us.zip
42. US Bureau of Labor Statistics, Quarterly Census of Employment and Wages, 2025 Q4, NAICS 8111, US, private ownership: 175,926 establishments (retrieved 2026-10-14). data.bls.gov/cew/data/api/2025/4/industry/8111.csv

Working draft. Version history: v0.14 (the open questions decided and the answers built: a confirmed meeting changes only with every member's yes, a café can be a meeting member, the `meetings` capability, members shown by your own contact names, hosts state how long they keep records, receipts say how the person agreed and under which mandate, short consent guidance for assistants; recurring meetings not in v0.4 and finding a place out of scope; SchCP repositioned as its own protocol rather than a profile of the business-booking efforts; closed schemas for holds, waitlist entries and booking responses; the v0.2 and v0.3 review probes retired, §10.22); v0.13 (tested live with AI assistants on both sides for 25 rounds, until three clean rounds in a row; a host that approves each booking approves a guest's move too; per-member answers; `resource_name` in offers; a yes may be taken back before a meeting is final, and backing out after it calls the meeting off; a business's own answer is `by: "host"`; scope narrowed to scheduling people, with entitlements, sales and searching across a brand's locations left out; “wallet” renamed the key app and “budget” a shared booking limit, since neither holds money, and a cancelled booking now gives its unit back; 22 conformance vectors for a guest's move and closed `Booking` and `ReceiptClaims` schemas; four new MCP tools and many wording fixes, §10.21); v0.12 (late cancellations and no-shows left to the host: no fee terms in offers, nothing recorded, no undo or cancel quote; attendance is still marked, for standing; no compatibility shims, since nothing has been released, §10.20); v0.11 (meetings of several people as one contract every member accepts by its hash; a host's yes from its person or from a rule the person signed, with the person strongly preferred; counters to a named place; a playdate for three families; travel time and per-staff skills out of scope by design, §10.19). Earlier: v0.1 (the problem statement and sketch); v0.2 (prior-art survey and expanded design); v0.3 (reference implementation and build report); v0.4 (closed mandates, attestations and decaying reputation, built and tested); v0.5 (ten business-viewpoint reviews, four defects fixed, an abstract and conclusion rewritten around the ask, AdCP-style licensing); v0.6 (a deep review's 20 defects fixed, a decision brief and contribution matrix, external claims corrected and linked, an operating example and a narrower ask); v0.7 (protocol v0.3 built: hashed contracts on every acceptance, all-in totals, recorded outcomes, undo, waitlist, two-party confirmation and refusal reasons); v0.8 (two reviews of v0.3 and their 22 fixes, scenarios for a mechanic shop, a sales meeting and colleagues, and narrower claims); v0.9 (every reviewer gap with a scenario closed, crash tests and the defect they found, conformance vectors and runner, an MCP server and an AP2 profile); v0.10 (protocol v0.4: SchCP never moves money; retitled around arranging people's time; hosts may be businesses or people, with contacts-only calendars; open-ended and unpriced bookings, named places, and a playdate between two families; the payment machinery and its tests removed, §10.18). The reference implementation, spec and research live in the same repository as this paper, with the diagrams in `docs/diagrams/` and the Remotion source of a 131-second explainer video in `video/`.