Architectural Choices

> Project: E2EE Messaging Platform
> Date: 2025-12-07
> Scope: Current architecture, rationale, and known trade-offs reflected in code, Docker/K8s manifests, and CI.

---

1) Architecture Style: Microservices with a Thin Gateway

Decision
Keep a small set of Go services behind an HTTP/WS gateway.

Services (current)

- **Gateway:** Reverse-proxy for HTTP + WS, CORS, rate limiting, request/trace IDs.
- **Auth:** Users, credentials, sessions, JWT mint/verify, device registry.
- **Keys:** Stores/serves identity keys, signed prekeys, and one-time prekeys.
- **Messages:** Stores ciphertext envelopes and delivers them via HTTP + WS fan-out.
- **Crypto-core (library):** Signal-style X3DH + Double Ratchet implementation consumed by the frontend.

Motivation

- Keep critical concerns isolated (auth, key directory, message queue/delivery).
- Swap/scale services independently; gateway hides service layout from clients.
- Crypto primitives live in a tested library, not the gateway/services.

Not currently planned

- File service and event bus mentioned earlier are not implemented yet; focus is on core messaging + key flows first.

Consequences

- More moving pieces than a monolith, but each service is small and purpose-built.
- Gateway must stay aligned with downstream routes and auth configuration (HS256 shared secret vs JWKS).

---

2) Language & Runtime: Go

Decision
All backend services and the crypto-core library use Go 1.25.x.

Motivation

- Strong stdlib for crypto, HTTP, and testing.
- Static binaries and small images; easy Docker packaging.
- Shared language across services keeps tooling simple (golangci-lint, go test).

Consequences

- GORM used for velocity; migrations managed explicitly with migrate containers.
- Frontend TypeScript bridges to the Go crypto-core via a local TS wrapper.

---

3) Security Model: Client-Owned E2EE

Decision
Encryption happens on the client using the `crypto-core` library (X3DH handshake + Double Ratchet). Servers see only ciphertext and ratchet headers.

How it works now

- Devices publish identity key + signature key, signed prekey, and a batch of one-time prekeys via **Keys**.
- Senders fetch prekey bundles, run X3DH, and start the ratchet; headers carry ratchet state, not plaintext.
- **Messages** only persists `ciphertext` bytes + opaque `header` JSONB plus routing metadata.
- One-time prekeys are consumed atomically in the Keys service to prevent reuse.

Consequences

- Servers cannot decrypt content; debugging relies on metadata and logs.
- Attachment flows are not built yet; would follow the same client-encrypt-first rule.
- Protocol-level work lives in `services/crypto-core` with deterministic and fuzz tests to catch regressions.

---

4) Authentication & Sessions

Decision
HS256-signed JWTs (access + refresh) with server-side session rows; optional device binding.

Details

- Auth service creates a `sessions` row (IP + user agent normalized) and mints access/refresh tokens.
- Access tokens carry `sid` and optional `did` claims; `Verify` enforces device ownership when provided.
- Refresh rotates the DB `refresh_id`, extends expiry, and re-issues both tokens.
- Gateway can validate via JWKS (if configured) or shared HS256 secret; defaults to shared secret in Compose.

Consequences

- Predictable revocation via DB and refresh rotation.
- IP is stored without port using `netip` normalization to satisfy `inet` columns.
- Device-scoped tokens let the message/keys services block cross-user device reuse.

---

5) Data & Persistence

Decision
PostgreSQL per service (`authdb`, `keysdb`, `messagesdb`) with migrations run by dedicated migrate containers.

Schemas (high level)

- **Auth:** users, credentials, sessions (`inet` IP, user agent), devices, MFA/audit scaffolding.
- **Keys:** users/devices plus identity keys, signed prekeys, and consumable one-time prekeys.
- **Messages:** append-only message table with ciphertext `BYTEA`, opaque `header JSONB`, sent/received/delivered timestamps.

Motivation

- Per-service DBs keep blast radius small and allow independent migration cadence.
- JSONB headers avoid server-side parsing of ratchet metadata.

Consequences

- GORM for runtime access; migrations kept in `services/*/migrations` and built into Docker images.
- No cross-service DB reads; everything goes through HTTP APIs.

---

6) Communication Patterns & Protocols

Decision

- HTTP/REST behind the gateway for auth, key, and message management.
- WebSockets proxied by the gateway for downstream message delivery.

Details

- Message sends are HTTP POSTs; delivery uses WS with periodic polling/ping from the Messages service to push pending envelopes and mark them delivered.
- Gateway adds CORS, rate limiting, request/trace IDs, and proxies `/auth`, `/keys`, `/messages`, `/ws`.
- Services call Auth for token verification; no internal message bus yet.

Consequences

- Simple client surface (single base URL) but gateway routing must stay in sync.
- Ordering/idempotency handled at the envelope level; no group semantics yet.

---

7) Containerization & Environments

Decision
Docker-first for dev and prod, with optional Kustomize manifests for Minikube.

Dev

- `.docker/docker-compose.dev.yml` brings up Postgres, run-once migrate jobs, auth/keys/messages/gateway.
- `.docker/docker-compose.observability.yml` optionally adds Prometheus, Loki/Promtail, and Grafana.

Prod

- `.docker/docker-compose.prod.yml` pulls GHCR images (`:latest` + `:<sha>`), runs migrations, and starts services.
- Secrets supplied via env (HS256 signing key, DB password); gateway/auth share the same secret when using HS256.

K8s

- `k8s/base` + `k8s/overlays/minikube-*` capture early Kustomize manifests; not the primary deployment path yet.

Consequences

- Migrations are part of the boot flow; bring-up fails fast if DBs or secrets are missing.
- Single-node Compose keeps ops simple until Kubernetes is needed.

7b) Local Minikube Deployment (Argo CD)

Decision
Run the platform on a local Minikube cluster via Argo CD using `infra/argocd/sem7-platform-minikube.yaml`.

Details

- Argo CD Application points at `k8s/overlays/minikube` on `main`; deploys auth/keys/messages/gateway.
- Argo CD Image Updater tracks GHCR images (`auth`, `keys`, `messages`, `gateway`) with the `latest` strategy and writes back to `main`.
- Destination is the in-cluster API (`kubernetes.default.svc`) and defaults to the `default` namespace; auto-prune + self-heal enabled.

Consequences

- Local cluster stays in sync with Git; drift is healed automatically.
- Image bumps are automated—keep `main` writable for updater commits.
- Requires Argo CD + Image Updater running in Minikube before applying the Application manifest.

---

8) CI/CD Strategy

Decision
Per-service GitHub Actions pipelines for lint, test, and image publish to GHCR; SonarQube on a self-hosted runner.

Workflows

- `lint.yml`: golangci-lint matrix across all Go modules.
- `*/.yaml` per service: go test, optional coverage upload to Sonar, build and push Docker images (`ghcr.io/klickk/secumsg-server/<service>`).
- Deployment using ArgoCD `7b`.

Consequences

- Fast, isolated feedback per service; crypto-core is tested as a first-class module.
- Requires GHCR access and Sonar tokens on CI.

---

9) Configuration & Secrets

Decision
Environment variables everywhere; Compose files wire them into containers. No secrets in git.

Details

- Auth signing key provided via `SIGNING_KEY` and reused by the gateway when validating HS256.
- Database URLs passed per service; prod Compose injects the password into each DSN.
- CORS origins set on the gateway; Auth issuer/audience configurable.
- JWKS path exposed by Auth if/when asymmetric keys are added.

Consequences

- Clear separation between code and secret material; easy rotation by re-running Compose with new env.
- Misalignment between gateway/Auth secrets immediately breaks auth (by design).

---

10) Observability & Logging

Decision
Structured slog logs plus Prometheus metrics and health endpoints across services; optional Loki/Grafana stack for dev.

Details

- `/healthz` and `/metrics` on every service; request/trace IDs injected by middleware.
- Message store/delivery metrics (counts, ciphertext sizes), auth token issuance metrics, gateway Prom metrics.
- `.docker/docker-compose.observability.yml` wires Prometheus + Loki/Promtail + Grafana dashboards.

Consequences

- Works locally with Compose; can be pointed at remote monitoring later.
- No distributed tracing backend yet; trace IDs are logged for future correlation.

---

11) Testing Strategy

Decision
Unit tests per Go module; crypto-core has deterministic + fuzz coverage for protocol safety.

Current state

- Auth: service-level tests for auth/device flows.
- Keys: service tests for bundle registration/rotation and OTK consumption.
- Crypto-core: protocol tests exercising X3DH + Double Ratchet, deterministic vectors, and fuzz harness.
- Messages: covered by store/service tests for enqueue/history basics; WS path exercised manually.

Gaps

- No end-to-end integration tests across services yet.
- No contract tests between gateway ↔ downstream or frontend ↔ gateway.

---

12) Risks & Mitigations

- **Protocol regressions:** Mitigated by crypto-core fuzz/deterministic tests; still need E2E integration.
- **Auth/config drift:** Shared HS256 secret between Auth and gateway is a single point of failure; JWKS/offline rotation planned.
- **Operational drift:** Compose prod lacks automation; add CI deploy or kube manifests once environments stabilize.
- **Limited observability:** Metrics/logs exist, but no tracing backend; add later when services grow.
- **Schema evolution:** Migrations are explicit; keep backward-compatible changes when rolling updates start.

Appendix: Component Responsibilities (Current)

- **Gateway:** CORS, rate limiting, request/trace IDs, proxy to Auth/Keys/Messages, WS pass-through.
- **Auth:** Users/credentials, device registry, HS256 JWT issuance/verify, session persistence (`inet` IP, UA).
- **Keys:** Key directory for identity/signed prekeys + one-time prekeys; validates token/device ownership on reads/writes.
- **Messages:** Ciphertext envelope store, HTTP send API, WS delivery loop with batch polling + delivery marking.
- **Crypto-core (library):** X3DH handshake + Double Ratchet, keypair/prekey generation, deterministic/fuzzed tests.
- **PostgreSQL (per service):** Auth/key/message data isolation; migrations via migrate containers.
- **Observability stack (dev optional):** Prometheus, Loki/Promtail, Grafana wired via Compose.
