Firelink Relay

Architecture & Security White Paper

Version 1.0 — April 2026 https://firelinkrelay.com

Executive Summary

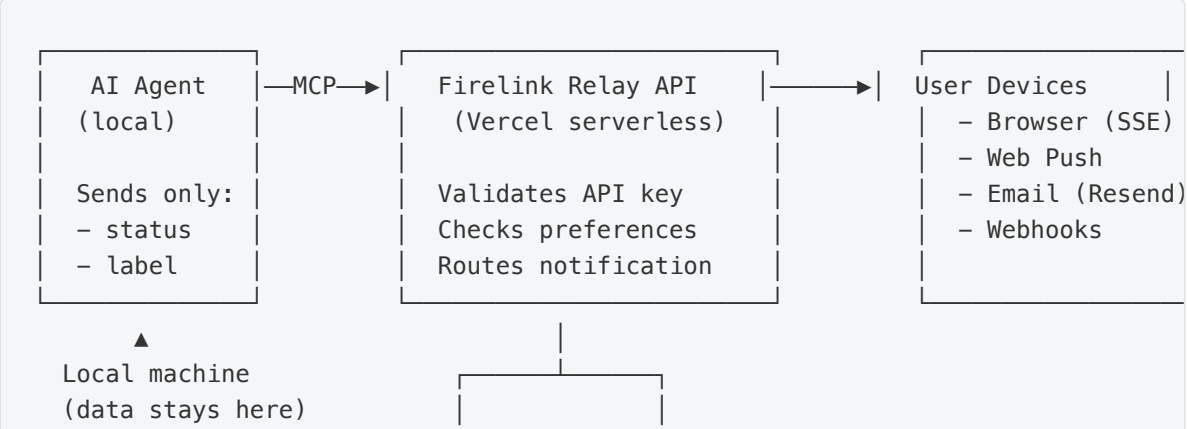
Firelink Relay is a notification relay service that lets AI agents (Claude, GPT, Gemini, Codex, and others) notify users when tasks complete or require human attention. It operates as a **signals-only layer** — transmitting only predefined status codes and user-supplied labels. By design, no AI conversation content, files, prompts, or private data ever reaches Firelink Relay servers.

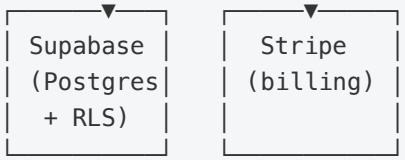
This document describes the system architecture, data model, security controls, and trust model that underpin this privacy guarantee.

---

1. Architecture Overview

Firelink Relay follows a simple three-tier architecture: an AI agent sends a signal via the Model Context Protocol (MCP), the Firelink Relay server processes and routes it, and the user receives the notification through one or more delivery channels.





Communication Protocol

Firelink Relay implements the **Model Context Protocol (MCP)** via Streamable HTTP at a single endpoint (`POST /api/mcp`). MCP is an open standard for AI agent tool integration. The server exposes one tool:

TOOL	PARAMETERS	DESCRIPTION
`send_notification`	`label` (string), `status` (`jobDone` or `needsUserInteraction`)	Sends a notification to the user

The `label` is a short, user-facing description (e.g., "PR review complete"). The `status` is one of two predefined values. No other data is accepted or transmitted.

Delivery Channels

When a notification is created, it is delivered in parallel through all enabled channels:

1. **Server-Sent Events (SSE)** — Real-time push to the browser dashboard, with in-memory connection management and automatic reconnection. SSE connections have a maximum lifetime of 4 minutes to control serverless compute costs.
2. **Web Push** — Native browser push notifications via the Web Push API (RFC 8291 / RFC 8292), allowing alerts even when the browser tab is closed.
3. **Email** — Per-notification transactional emails via Resend, opt-in with a rate limit of 10 emails per user per hour.
4. **Webhooks** — Outbound HTTP POST to user-configured endpoints (Discord, Slack, or custom URLs) with HMAC-SHA256 signed payloads.

---

2. Data Minimalism

Firelink Relay is architected around a core principle: **collect and store the absolute minimum data required to deliver notifications.**

What We Store

CATEGORY	DATA	PURPOSE
Account identifier	OAuth provider user ID	Link preferences to account
Notification metadata	Status type + user-supplied label	Display the notification
API key hashes	SHA-256 digest (never the raw key)	Authenticate MCP requests
Webhook destinations	URLs + HMAC signing secrets	Deliver outbound webhooks
Push subscriptions	Web Push endpoint + encryption keys	Deliver push notifications
Stripe customer ID	Stripe-issued identifier	Link billing subscription

What We Never Store

CATEGORY	REASON
AI conversation content, prompts, or responses	Never sent to us — agent runs locally
Files, code, or documents	Never sent to us
Credit card numbers or payment details	Handled entirely by Stripe
Workflow patterns or usage analytics	Not collected
IP addresses or browser fingerprints	Not logged

The key insight is architectural: because Firelink Relay sits *outside* the AI agent's execution environment and only receives two predefined status signals, there is no mechanism by which private data could be transmitted to our servers — even accidentally.

3. Security Controls

3.1 API Key Security

API keys authenticate MCP requests from AI agents to the Firelink Relay server.

- **Hashing**: Keys are hashed with SHA-256 before storage. The raw key is shown once at creation and never stored or retrievable.
- **Entropy**: Each key contains 32 bytes of cryptographically random data (`crypto.randomBytes(32)`).
- **Rate limiting**: Requests are capped at 30 per minute per API key, enforced at the application layer.
- **Audit trail**: A `last_used` timestamp is updated on each successful authentication for anomaly detection.
- **Revocation**: Users can revoke keys instantly from the dashboard. Revoked keys are deleted from the database.

3.2 Database Security & User Data Isolation

All data is stored in PostgreSQL (hosted by Supabase) with Row-Level Security (RLS) enforced at the database engine level.

- **Row-Level Security (RLS)**: Every table — `profiles`, `api_keys`, `notification_prefs`, `notifications`, `push_subscriptions`, and `webhook_destinations` — has RLS policies enabled. Each policy restricts row access to the authenticated user's own data using `auth.jwt()->>'sub'`. This means User A's notifications, API keys, and preferences are completely invisible to User B at the database engine level.
- **Defense in depth**: Because RLS is enforced by PostgreSQL itself (not the application layer), even a complete application compromise — such as a code injection or logic bug — cannot leak data across user boundaries. The database rejects unauthorized queries before they execute.
- **User-scoped JWTs**: Each authenticated request carries a JWT containing the user's identity. The database extracts the user ID from this token and applies it to every query

automatically. There is no way to forge or omit this identity check.

- **Service-role separation**: A small number of administrative operations (e.g., incrementing the monthly usage counter, cleaning up expired push subscriptions) use a separate service-role client. This client bypasses RLS by design but is restricted to server-side code only — it is never exposed to end users or client-side requests.
- **No shared tables**: There are no multi-tenant tables where users share rows. Every row belongs to exactly one user, enforced by foreign key constraints and RLS policies.

3.3 Webhook Security

Users can configure up to 5 outbound webhook destinations. Each webhook delivery includes:

- **HMAC-SHA256 signatures**: Every outbound payload is signed with a per-webhook secret using HMAC-SHA256. The signature is included in the `X-Firelinkrelay-Signature` header. Recipients can verify payload authenticity by recomputing the HMAC.
- **Timestamp header**: An `X-Firelinkrelay-Timestamp` header is included for replay attack protection. Recipients should reject payloads with timestamps older than a reasonable window.
- **HTTPS-only enforcement**: Webhook URLs must use the `https://` protocol. HTTP URLs are rejected at configuration time.
- **SSRF protection**: Before delivering a webhook, the destination hostname is resolved via DNS and checked against a blocklist of private/internal IP ranges (127.0.0.0/8, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16, IPv6 ULA, link-local, and cloud metadata endpoints). This prevents Server-Side Request Forgery (SSRF) attacks.
- **Timeout**: Webhook deliveries have a 5-second timeout. Slow or unresponsive endpoints do not block notification processing.

3.4 Push Notification Security

Web Push notifications follow the IETF standards for encrypted push messaging:

- **Payload encryption**: AES-128-GCM as specified in RFC 8291 (Message Encryption for Web Push).

- **Server authentication**: VAPID (Voluntary Application Server Identification) as specified in RFC 8292. The server signs each push message with its VAPID private key, proving the message origin.
- **Endpoint-scoped keys**: Each browser subscription generates unique P-256 encryption keys. The server can only send to subscriptions it holds keys for.
- **Automatic cleanup**: Subscriptions that return HTTP 410 (Gone) or 404 (Not Found) are automatically removed from the database.

3.5 Email Security

Email notifications are delivered via Resend with the following controls:

- **Rate limiting**: A per-user cap of 10 emails per hour prevents quota exhaustion and abuse.
- **Opt-in only**: Email notifications are disabled by default. Users must explicitly enable them in settings.
- **Domain authentication**: Emails are sent from `noreply@firelinkrelay.com` with SPF and DKIM records configured for deliverability and anti-spoofing.
- **Graceful degradation**: If the Resend API key is not configured, email delivery silently no-ops without affecting other channels.

3.6 Authentication & Billing Isolation

- **Authentication**: Handled by Clerk, a dedicated identity provider supporting OAuth via Google, Apple, and GitHub. Firelink Relay never handles or stores passwords.
- **Billing**: Handled entirely by Stripe. Credit card numbers, bank details, and payment methods never touch Firelink Relay servers. Only a Stripe-issued `customer_id` is stored to link a user's subscription status.
- **Subscription verification**: Subscription status is checked live against the Stripe API (with a short in-memory cache) on each MCP request. No billing state is persisted in the application database.

---

4. Data Flow Analysis

What Crosses the Network

To make the privacy guarantee concrete, here is an exhaustive list of data that travels from the AI agent's environment to Firelink Relay:

1. **API key** — Sent in the `Authorization: Bearer` header. Used for authentication, then discarded (only the hash is compared).
2. **Status** — One of two string values: `jobDone` or `needsUserInteraction`. No custom values accepted.
3. **Label** — A short text string the agent provides (e.g., "Build completed"). This is the only user-controlled content that reaches the server.

That is the complete set. No conversation history, no file contents, no prompts, no model outputs, no environment variables, no system information.

What Stays Local

The AI agent's execution environment — including conversation context, file system access, code, credentials, and all model interactions — remains entirely on the user's local machine or their own cloud infrastructure. Firelink Relay has no mechanism to request, receive, or infer this data.

---

5. Infrastructure

Deployment

Firelink Relay runs on **Vercel** as a serverless application. Each API request is handled by an isolated serverless function with:

- No persistent disk access
- No shared memory between invocations (except in-memory caches within a single instance lifetime)
- Automatic scaling and geographic distribution
- HTTPS-only with automatic TLS certificate management

Database

Supabase (managed PostgreSQL) provides:

- Encryption at rest (AES-256)
- Encryption in transit (TLS 1.2+)
- Automated daily backups
- Point-in-time recovery
- Network isolation (database not publicly accessible without valid credentials)

Third-Party Services

SERVICE	PURPOSE	DATA SHARED
Clerk	Authentication	OAuth tokens (managed by Clerk)
Stripe	Billing	Customer ID only (no payment data on our servers)
Resend	Email delivery	Recipient email + notification content
Supabase	Database	All application data (encrypted at rest)
Vercel	Hosting	Application code + runtime logs

---

6. Trust Model

Firelink Relay's trust model is built on **architectural transparency** rather than security through obscurity.

Design Principles

1. **Data minimalism by design** — We physically cannot access data we never receive. The architecture ensures private data never leaves the user's machine.
2. **Standard protocols** — MCP, RFC 8291/8292 (Web Push encryption), HMAC-SHA256 (webhook signing). No proprietary cryptographic implementations.

3. **Database-level enforcement** — PostgreSQL Row-Level Security policies enforce data isolation regardless of application bugs. Even a complete application compromise cannot leak data across user boundaries.
4. **Third-party payment handling** — Stripe processes all payment data. PCI compliance is delegated to a certified payment processor.
5. **Responsible disclosure** — Security vulnerabilities can be reported via the contact form at `https://firelinkrelay.com/support` or through the `/.well-known/security.txt` endpoint.

Limitations & Transparency

- **Label content**: The notification label is user-controlled free text. Users should avoid including sensitive information in labels, as labels are stored in the database and may be transmitted to webhook destinations.
- **In-memory state**: SSE connections and rate-limit counters are held in-memory on serverless instances. This state is ephemeral and does not persist across function invocations.
- **Webhook destinations**: When users configure webhooks, notification payloads are sent to those external URLs. Firelink Relay signs the payloads but cannot control how third-party services handle the data.

---

7. Rate Limiting & Abuse Prevention

CONTROL	LIMIT	SCOPE
MCP API requests	30/minute	Per API key
Free tier notifications	50/month	Per user
Email notifications	10/hour	Per user
Webhook destinations	5 max	Per user
SSE connections	10 max	Per user

SSE connection lifetime	4 minutes	Per connection
Webhook delivery timeout	5 seconds	Per request

---

8. Compliance Posture

Current State

- HTTPS everywhere (enforced by Vercel)
- No PII collection beyond OAuth account identifiers
- No tracking cookies or analytics
- GDPR-aligned data minimalism
- Payment processing delegated to PCI DSS Level 1 certified provider (Stripe)

Roadmap

- Third-party security audit (e.g., Trail of Bits, Cure53, NCC Group)
- Bug bounty program (HackerOne or Bugcrowd)
- SOC 2 Type II compliance process
- Content Security Policy (CSP) header implementation
- Strict Transport Security (HSTS) header
- Automated dependency vulnerability scanning

Contact

- **Website**: https://firelinkrelay.com
- **Security reports**: https://firelinkrelay.com/support (select "Technical Issue")
- **Security policy**: https://firelinkrelay.com/.well-known/security.txt

---

This document describes the architecture and security posture of Firelink Relay as of April 2026. It will be updated as the system evolves.