Security and privacy

Encryption at rest, encryption in transit, access controls, retention windows, AI-provider posture, the deliberate gaps we are transparent about, and how to report a security issue.

Up to date as of v1.32.0

CONTENTS

- What this chapter covers
- Encryption at rest
- Encryption in transit
- Tenant isolation
- Roles, seat types, and per-module access
- Login security and session handling
- Append-only audit ledgers
- Service Agreement encryption
- Transfer Room encryption
- AI providers — what gets sent and what does not
- Fax — the deliberate plaintext window
- Data retention
- Account deletion and the 90-day purge
- Where we draw the line — deliberate gaps
- Google Cross-Account Protection (RISC)
- Reporting a security issue
- External disclosures and legal documents

What this chapter covers

Security on RCIC App is layered. The platform protects your tenant + your client data with disk-level encryption underneath, application-level envelope encryption on top of the most sensitive modules (Service Agreements, Transfer Room, Active File Review, Co-Counselling Agreements, Booking Notes, Information Card data, Matter Work Ledger descriptions, Intake Forms responses), and tenant isolation enforced at every database query plus every API route. This chapter walks through the architecture deliberately so you can explain it to your clients, your accountant, or your insurer when they ask. The canonical public version of this disclosure lives at

rcicapp.ca/security and is linked from the marketing footer; the dashboard chapter you are reading mirrors that page and adds extra detail on operational practices that are not in scope for the marketing version.

Encryption at rest

Two layers protect your data when it sits in storage. The first layer is infrastructure-level disk encryption: the database (Supabase Postgres) and the object storage that holds every uploaded file are encrypted at the volume layer using AES-256. This is automatic for every Supabase project and protects against the simplest physical threats — someone walking out of a data centre with a hard drive cannot read it without the volume key.

The second layer is application-level envelope encryption that the platform writes on top of the first layer for sensitive client data. Service Agreements, Transfer Room files, Active File Review documents, Co-Counselling Agreements, Booking Notes descriptions, Information Card payloads, Matter Work Ledger entry text, and Intake Forms responses all get encrypted with AES-256-GCM under a per-tenant data-encryption key (DEK) before the bytes are written to either database column or storage object.

Each per-tenant DEK is itself wrapped under a master key (KEK) that lives in the application platform's environment, NOT in the database. The separation is deliberate: decrypting any sensitive field requires database access AND access to the master key, which are separate credentials held in separate systems with separate access trails. A database export, a Supabase backup tape, or a compromised database connection alone cannot reveal the protected client data; the master key would also have to leak. Rotating the master key is a single operator-level action that does not require touching individual tenant DEKs (the KEK rotation re-wraps every DEK in place).

Encryption in transit

Every connection between your browser, the application, the database, the payment processor (Stripe), the email provider (SiteGround SMTP for platform mail + rcicapp.ca SMTP for client-facing mail), the calendar providers

(Google + Microsoft 365), the cloud-translation provider (Google Cloud Translation), the AI providers (Google Gemini + Anthropic Claude), the fax carrier (Telnyx), and the Drive / OneDrive integrations is encrypted with TLS 1.2 or higher. The platform does not accept plain-HTTP traffic on any production endpoint; the HTTP-to-HTTPS redirect at the CDN layer enforces this even if a misconfigured client tries to bypass it. Webhook callbacks from third parties (Stripe, Telnyx, RISC, etc.) are similarly HTTPS-only and additionally carry signed payloads the platform verifies before processing.

Tenant isolation

Every database query in the application is scoped by company_id. The query plan always carries a company_id predicate, and the company_id is always resolved from the authenticated user's session — never from a URL parameter or a request body the caller controls. Postgres Row Level Security policies on every multi-tenant table are a belt-and-suspenders backstop: even a hypothetical query that forgot to add the company_id predicate would be filtered out by RLS before any row is returned. Cross-tenant access attempts at the API layer return 404 (not 403) deliberately — the platform does not even confirm whether a target row exists in another tenant, because confirming existence is itself an information leak. The pattern is uniform across every module on the platform.

Roles, seat types, and per-module access

Within a tenant, two role tiers stack on top of two seat types. The Owner role controls billing, team management, account deletion, and per-member module access; only one user holds the Owner role per tenant and the role cannot be transferred to a team member without a deliberate Owner-initiated action. Members hold one of two seat types: Assistant or RCIC. Default per-module access varies by seat type (RCIC seats default to access for every module; Assistant seats default to access for the operational modules but not the regulated ones — Service Agreements and Written Consultations specifically default to off for Assistants because the work is regulated immigration advice). The Owner can override the default on any member's row from Settings ’ Team ’ that member ’ Module Access.

Sensitive operations that go beyond per-module access — Owner-gated specifically — include account deletion, signed-PDF download, V2 subscription management, team credential changes (email and password rotation for any team member), and store top-ups. Assistants can prepare Service Agreements but cannot sign them; only verified RCICs (eligibility granted by the Owner, licence self-attested by the member through their /dashboard/profile page) can be named on agreements and sign them. The platform server-side validates every signing request against the RCIC attestation state at request time, not at form-render time — a stale 'Sign' button on a member's screen who has since lost RCIC status will return 403 on click.

Login security and session handling

Every login on a new device requires a one-time email code (OTP). The platform never accepts password-only authentication on a new device. Trusted devices are remembered for 30 days via a signed cookie; a thirty-first-day login on the same device re-prompts for OTP. Authentication is rate-limited per IP and per email address — a few failed OTP attempts in quick succession returns a backoff with a clear error, and persistent attempts trigger a temporary block. Passwords are stored as Argon2 hashes by Supabase Auth; the platform never receives or stores plaintext passwords. Password resets fire from /login ’ Reset password and require email-based confirmation.

Owners can also fire a password-reset email for any team member from Settings ’ Team without needing the member's permission.

Active sessions have an 18-minute idle timeout. A modal warning appears two minutes before the timeout fires, giving you a chance to extend the session by interacting with the page. The timeout is server-enforced via an HMAC-signed activity cookie checked in middleware on every dashboard request; closing your laptop without logging out and reopening it 30 minutes later requires a fresh login regardless of whether the browser tab remained open. Browser-tab coordination via BroadcastChannel ensures that a session timeout in one tab logs out every other tab for the same user simultaneously.

Append-only audit ledgers

Five modules carry append-only audit ledger tables that are technically protected against UPDATE — even the platform's service-role database client cannot modify a row once it has been written. The protection is a database trigger that raises an exception on any UPDATE attempt, including from administrative tooling. The ledgers cover Service Agreement signature events (who signed, when, from what IP, the typed name, the token hash prefix), Transfer Room events (room activated, transfer sent, viewed, downloaded, revoked, deleted, participant added or removed), Active File Review events (pathway accepted, payment received, deliverable uploaded, case closed), Co-Counselling Agreement events (the parallel set), Service Proposals events (sent, viewed, accepted, counter-offered, declined). Together the ledgers form a tamper-evident audit trail your insurer, the regulator, or a future legal action can rely on to reconstruct who did what when. DELETE is permitted on each ledger only as a cascade from the parent row (e.g. tenant account deletion); user-initiated direct DELETE is blocked the same way UPDATE is.

Service Agreement encryption

Service Agreements were the first module to ship application-level envelope encryption (Phase Encryption-1) and remain the most thoroughly covered. Every sensitive field on a Service Agreement is encrypted at rest under the tenant's DEK: client name, client email, client phone, client address, the full draft document JSON, every signing party's name + email + phone + address, the signed PDF bytes, AI-uploaded source documents, the per-clause initials. Searchable email columns carry an additional HMAC blind-index column (so a 'find this agreement by client email' lookup works without decryption) computed under a separate SEARCH_INDEX_KEY held outside the database. The legacy plaintext columns that used to hold this data were dropped from the database in a sunset migration; direct SQL access from a Supabase admin can no longer reach the protected fields in plaintext form.

Signed PDFs deserve a specific mention. After multi-party signing completes, the rendered PDF is encrypted under the tenant DEK before being uploaded to the agreement-pdfs private storage bucket (the storage path ends in .pdf.enc to make the encrypted-bytes posture explicit). The PDF is additionally owner-password protected via @cantoo/pdf-lib using a random 32-byte hex owner password generated per render and discarded after the upload — print, copy, and accessibility permissions are allowed; modify, annotate, fill-forms, and assembly are blocked. The combination means even an attacker with both database access AND the master KEK would only recover an owner-password-locked PDF rather than a freely-editable one. Email attachments of the signed PDF that go to the client are sent in plaintext (the recipient needs to be able to open them); the encryption applies to the platform's stored copy only.

Transfer Room encryption

Every file you or your client upload through Transfer Room is encrypted under your per-tenant DEK before it lands in the transfer-room private storage bucket. The same DEK protects every other encryption-aware surface in your tenant; cross-tenant decryption is mathematically impossible (a different tenant's DEK would not decrypt your file even with full database access). Clients sign in to the portal with their own account: the email address on file and

a password they choose, with optional two-step sign-in. Email sign-in codes are no longer used; a client who asks for one is emailed a sign-in or setup link instead, at the address on file.

Transfer Room files live for the period their sender chooses; the platform is not a long-term file vault. Each transfer expires on its sender's date: 30 days by default, anywhere from 1 to 90. Its files stay available until then, whether or not they have been downloaded or copied to your Drive, and are deleted at expiry. If your firm has a storage destination set up, a file is not deleted until a copy has reached it. Files in a revoked transfer are deleted 72 hours after revocation. The durable copy lives in your matter file or your Drive; the Transfer Room is the exchange channel, not the storage. One small architectural acknowledgement we are transparent about: between the moment a browser uploads a file to the signed-URL storage destination AND the moment the platform server downloads it, encrypts it under your DEK, and overwrites the original, the file bytes sit unencrypted in the private storage bucket for a few seconds. The bucket is not publicly readable and no API route serves those bytes; the only way to read them in that brief window would be direct administrative access to the storage layer. Closing this gap with client-side pre-encryption is on the roadmap.

AI providers — what gets sent and what does not

AI surfaces send content to a third-party model provider — there is no way around that for a cloud-LLM workflow. The platform uses two providers: Anthropic Claude for every Service Agreement AI surface (AI Populate extract + AI Review), and Google Gemini for everything else (AI Writing Assistant, booking AI summary, AFR intake triage, AFR + Transfer Room ID extraction, Matter Work Ledger code suggester, help-center chat, and the shared SA-extractor reuse from CCA / Service Proposals / Booking Notes). Both providers contract with the platform under enterprise-grade data-handling terms that the platform commercial agreement tracks: API content is not used to train provider models, and provider-side retention is bounded by the contract. Cloud Translation calls for runtime multilingual booking page + client-side SA translation go to Google Cloud Translation, which is under the same Google data-handling contract.

None of this should be confused with end-to-end encryption. The content is decrypted on the platform server before being sent to the AI provider, and decrypted on the provider server to be processed by the model — the model sees the content in plaintext for the duration of the API call. The platform does NOT train any model on your data, share content with other tenants, retain AI request bodies beyond the time needed to log the call against the daily quota, or send AI content to any provider beyond the two named. What the platform DOES retain is a per-call usage record (date + tenant + cost in quota slots or store points + success-or-failure outcome) — this is what the quota counter and the per-tenant reports surface read from. The /security marketing page carries the canonical disclosure of what gets sent where; refer to it when a client or counsel asks about AI use on their matter.

Fax — the deliberate plaintext window

The Fax module is the one platform surface where the security posture is genuinely different. Faxes travel over the public switched telephone network (PSTN), which does not encrypt fax transport — no fax provider, including ours (Telnyx), can change that. Confidentiality of an in-flight fax depends on the receiving machine and the people who physically have access to it. The platform discloses this in a versioned consent screen tenants acknowledge before their first send and before opting in to receive. Source PDFs that tenants upload for outbound sends transit a fax-source-temp storage bucket in plaintext for the duration of the send (typically seconds; orphans are swept hourly); outbound bytes never persist past the send. Inbound bytes encrypt at rest under the receiving tenant's per-tenant data-encryption key — the same key Transfer Room documents use — once the routing decision lands, but for the brief window between Telnyx delivery and the platform's encrypted-at-rest write, the bytes sit unencrypted in the platform's fax-receive bucket.

When an inbound fax cannot be auto-routed (the cover page lacks a routing QR or a readable Tenant Number), the fax lands in a SaaS-admin quarantine review queue. The platform's superadmin can open the fax to decide which tenant it belongs to; once assigned, the fax encrypts at rest under the receiving tenant's key. Quarantined

faxes stay accessible to the SaaS administrator until they are assigned or rejected. Rejected, assigned, and still-quarantined rows older than 90 days are purged daily by a cron job. The receive-side consent screen explicitly discloses this SaaS-admin access pattern — by opting in to receive, the tenant accepts that the operator may open a quarantined inbound fax to make a routing decision.

Data retention

- Active tenants — data retained for the life of the account. There is no auto-expiry on any tenant-owned record while the account is active.
- Deleted accounts — soft-deleted immediately on the Owner-initiated delete action, then permanently purged 90 days later by an automated cron job. Storage objects (signed PDFs, uploaded source documents, Transfer Room files, AFR documents, intake responses, fax bytes) are removed in the same purge wave. The 90-day window exists so accidental deletions can be recovered via platform support during that period.
- Signed Service Agreements — retained for the duration of your account so you can download them as legal records. CICC professional retention obligations are your responsibility as the licensee; the platform surfaces this reminder when you initiate account deletion so you can take a backup before the data purges.
- Email logs — SMTP delivery records (success / failure timestamps, recipient address) retained 90 days for deliverability analysis. Email body content is not stored on the platform's servers after delivery.
- Audit ledgers — retained for the duration of the parent record (Service Agreement, Transfer Room, Active File Review, etc.); cascade-deleted only when the parent is hard-purged at the 90-day post-account-deletion mark.
- Webhook delivery logs — retained 30 days for debugging. Webhook payload bodies are not stored beyond the time needed to process and acknowledge the delivery.

Account deletion and the 90-day purge

Account deletion is Owner-only and irreversible after the 90-day grace window. From Settings ’ Account ’ Delete account, the Owner is presented with a typed-confirmation prompt that requires typing the tenant slug to confirm. On confirmation, the platform sweeps the Store wallet through a closure-refund calculation (paid points refunded at 95% to the original card; bonus / admin-grant points forfeited), soft-deletes the tenant, fires a closure email to the Owner, and schedules the hard purge for 90 days from now. During the 90-day window the tenant is unrecoverable through self-serve but can be restored by platform support if the deletion was accidental.

At day 90, a daily cron sweeps soft-deleted tenants whose grace window has expired. The cron walks every related table by company_id and hard-deletes the rows (cascade-deleting audit ledgers, signature events, etc.), removes every storage object owned by the tenant from the Supabase Storage buckets, and deletes the auth.users rows for every team member who was a member only of this tenant (multi-tenant users keep their auth row). After the purge runs, nothing about the tenant is recoverable through any technical path on the platform — the only durable copies are whatever you (the Owner) exported or downloaded before deletion, and the records held by external systems (Stripe, your connected calendar, your Drive). The hard-purge process completes within minutes; the deletion of fast-moving auth rows happens immediately.

Where we draw the line — deliberate gaps

Three architectural gaps in the current encryption posture are worth being explicit about.

- Not end-to-end. The application server holds the master encryption key in memory while serving requests. A determined and authorised administrator with both database access AND the application's environment access can decrypt your data; they would just leave a clear access trail across two separate systems. True end-to-end encryption (where only the tenant's browser holds the key) is a fundamentally different product shape and is not on the near-term roadmap.
- Some modules are not yet covered by application-level encryption. Booking + Event registration data (client name, email, phone associated with a booking row) remains in plaintext at the database column level (still

protected by disk-level encryption + tenant isolation). Bringing those modules under the same envelope-encryption posture as Service Agreements + Transfer Room is a roadmap item; the gap is acknowledged.

- Client-side device security is your responsibility. If your laptop, browser, or password is compromised, an attacker can sign in as you and the platform cannot tell the difference. Use a password manager, expect OTP on every new-device login, and treat the device-trust cookie the way you treat a house key — losing the trusted device means immediately changing your password and re-issuing OTP-trusted-device status.

Google Cross-Account Protection (RISC)

For tenants who connect their Google Drive (or Google Calendar) to the platform, Google's Cross-Account Protection (RISC) is wired in. When a connected Google account is disabled by Google, has its sessions or tokens revoked, has credential changes required, or is otherwise flagged for security action by Google, Google pushes a signed Security Event Token to a platform receiver endpoint. The receiver verifies the signature against Google's RISC JWKS, then acts on the event by clearing the Google account's Drive tokens on your tenant and flipping the connection status to 'reauth_required'. Your dashboard surfaces a 'Reconnect Google Drive' banner the next time you open the Drive settings page. The pattern means an account-takeover-style event on the Google side propagates protective state to the platform within seconds, even if you have not yet had a chance to react to Google's email.

Reporting a security issue

Found a vulnerability or have a security question that goes beyond what the in-app Report an Issue form is the right channel for? Email the platform team directly at info@investatech.com with 'Security Report' in the subject line. Security reports are treated as priority and the platform team aims to acknowledge within one business day. Please do not test against another tenant's data — if you find an issue that requires reproducing against a real cross-tenant boundary, describe the scenario in the email and the team will reproduce it internally with

synthetic data rather than against your own. The platform does not currently run a paid bug-bounty program; the response we offer is acknowledgement, prompt remediation, and a credit in the changelog if you would like one. For non-security platform bugs (something is broken, but no cross-tenant leak or data-handling concern), the in-app Report an Issue form is the right channel; see the dedicated chapter on reporting issues.

External disclosures and legal documents

Three external surfaces carry the canonical legal + security wording you can link clients, counsel, or insurers to. The /security marketing page at rcicapp.ca/security is the public version of this chapter; it is the document to share when a client asks 'where is your security policy written down'. The /privacy page covers personal-information handling under PIPEDA + Quebec Law 25 framings, including how the platform handles cookies, analytics, AI providers, and cross-border transfers. The /terms page covers the commercial relationship including limitations of liability + service-level posture + dispute resolution. All three are bilingual EN+fr-CA and linked from the marketing footer; they update through versioned redlines surfaced via the in-app policy-update notice email when material changes ship.