Secure PDF Tools

Eleven PDF utilities that run in your browser: Compress, Split, Merge, Convert to PDF, PDF to Images, OCR, Organize Pages, Protect, Stamp, Redact, and Sign PDFs. Ten run entirely on your device — files never leave your browser. The eleventh, Sign PDFs, is server-side by necessity (it coordinates signatures from multiple recipients across multiple sessions).

Up to date as of v1.30.0

CONTENTS

- Edit PDF — click-to-replace existing text
- Edit PDF — fourteen shared font families

Overview

Secure PDF Tools is the eleven-tool utility module that gives you the operations a typical immigration practice runs on PDFs every day. The headline promise is privacy: ten of the eleven tools run entirely in your browser through Web Workers and WebAssembly engines, so the file bytes never reach our servers, our storage, our backups, or our logs. You can use the module on a client's identity document, an IRCC refusal letter, a passport scan, or any other sensitive PDF with the same confidence you would have using an offline tool installed on your machine — minus the install, the licence-key admin, and the operating-system compatibility check.

The exception is Sign PDFs. Electronic signature coordination requires storing the file securely between the moment you upload it and the moment every recipient has signed, generating per-recipient tokens that resolve weeks or months later, applying signatures from multiple recipients in multiple sessions, building an audit certificate, and emailing the executed package to everyone with a tamper-evident PDF. That cannot live in a browser tab — your machine and tab need to be off when your client signs at 11 PM from their phone. The Sign tool runs server-side with the same encryption-at-rest posture as the rest of the platform (per-tenant DEK + master KEK), and the upload form spells out the difference up front.

> **Note**: Secure PDF Tools is open to every tenant on every tier. Basic and Premium both see the module, the hub, every tool, and the same operational behaviour. Nothing in this chapter is Premium-gated. The Save-to-Drive integration (Google Drive or Microsoft 365 OneDrive) is the only optional capability that requires a one-time OAuth connect; the connect happens elsewhere in the dashboard and once connected, every Secure PDF Tool can save its output directly to your Drive.

Your files stay on your device

The privacy posture for the ten client-side tools is unconditional. From the moment you pick a file in the picker to the moment you click Download, every byte stays inside your browser tab. The Web Worker that processes the file talks to the page through postMessage; the page talks to the disk through standard browser file APIs; nothing in that chain reaches the network. We do not stream telemetry, we do not log file names, we do not send fingerprints of the bytes, and we cannot recover your file if you close the tab before saving (because we never had a copy).

Two narrow exceptions are worth knowing about. Save-to-Drive (an optional output destination on every tool) sends the OUTPUT file to your own Google Drive or OneDrive folder; the original file still never leaves your device, but the output transits briefly through Supabase storage on its way to your Drive because the Vercel function that talks to Google's or Microsoft's API has a 4.5 MB request-body cap. We work around the cap with a signed-URL upload that puts the bytes directly into a tenant-scoped storage path, then the function downloads them from storage and forwards to your Drive. The temporary storage object is deleted immediately on success and a daily cron sweeps any orphans after one hour. The second exception is Sign PDFs, which lives server-side by design (see the dedicated section below).

> **Important**: We refuse to process encrypted PDFs. If a PDF has been password-protected by another tool (whether user-password or owner-password), the upload step in any of the ten client-side tools refuses the file with a clear message asking you to remove the protection first. The exception is the Protect tool, which exists to ADD protection to an already-unprotected PDF; it carries its own already-protected detection that catches the rare case where you try to protect a file twice. See the Encrypted PDFs are refused section below for the why.

The hub landing page

Clicking Secure PDF Tools in the sidebar lands you on /dashboard/secure-pdf-tools, the hub picker. The page carries a header with a green privacy banner that reads 'Your files stay on your device', a free-text search box that filters the eleven tiles in place as you type (by tool name or keyword like 'rotate', 'password', 'image'), and a responsive grid of eleven tiles — one per tool — with a short description below each tool name. Click a tile to open the tool. The sidebar mirrors the same set of entries, plus an All tools entry at the top that brings you back to the hub.

Each sub-tool lives at its own subroute: /compress, /split, /merge, /convert, /images, /ocr, /organize, /protect, /stamp, /redact, /sign. You can bookmark a specific tool if you use it often. The browser URL is the only state that lives outside the tab; closing and reopening the tool tab loses any in-progress work because nothing is persisted server-side (the privacy posture trades durability for confidentiality, deliberately).

Browser support

The module needs three modern browser capabilities to run: OffscreenCanvas (so Web Workers can render PDFs without a DOM), module Workers (so the engine code can use modern JavaScript module syntax), and SubtleCrypto (so we can hash output for the trust badge). The minimums are Chrome 91+, Microsoft Edge 91+, Firefox 105+, and Safari 16.4+. On older browsers, the tool workspace is replaced with a friendly amber 'Your browser does not support Secure PDF Tools' card that lists the minimum versions; nothing breaks silently. Roughly ninety-seven percent of practice-typical traffic already lands above these thresholds.

On screens under 768 pixels wide (most phones), each tool surfaces a dismissible amber mobile warning at the top. The tools do work on mobile, but the touch UX for tasks like dragging pages around in Merge Page Mode or Organize, or boxing redaction zones in Redact, is awkward compared with a mouse. The warning suggests using a laptop or desktop for the involved tools and clears for the rest of the session once dismissed.

Encrypted PDFs are refused

Every tool that ingests a PDF runs an encryption pre-flight check before processing. When the check detects a password-protected file (either a user password that prompts on open, or an owner password that allows reading but blocks editing), the tool refuses with a clear error: 'This PDF is protected or encrypted. Please remove the password before processing.' The refusal is intentional. The pdf-lib engine we use to mutate PDFs can read most encrypted PDFs but cannot rewrite their object stream back into the original encryption envelope without the password — silently dropping the protection would be a confidentiality violation; silently rewriting with a different protection would be a data-integrity violation.

Owner-password-only PDFs are a special hazard because they open without prompting and look unencrypted in most viewers; an early version of the module relied on substring matching against the error message returned by pdf-lib and missed owner-password-only files entirely (the error message says 'password' for user-password files and is silent for owner-password files). The current pre-flight runs a structural inspection of the PDF document object instead — it always catches both kinds, on every tool, every time.

The Protect tool is the one exception to the refuse-encrypted policy. Protect's job is to add a password to a PDF that does not already have one; it carries its own already-protected detection that fires with the friendlier 'This PDF is already password-protected' message instead of the generic 'protected or encrypted' message. Either way, the resolution is the same: open the file in Adobe Acrobat or a similar tool, remove the password using the document properties dialog, and re-upload the unprotected copy to whichever Secure PDF Tool you need.

XFA dynamic forms (IRCC IMM 0008 / 5257 / 5645)

A specific class of IRCC PDF forms uses Adobe LiveCycle XFA (XML Forms Architecture) — the modern IMM 0008 generic application, the IMM 5257 temporary resident visa, the IMM 5645 family information, and several others. These are not regular PDFs with fillable fields; they are XML documents wrapped in a PDF container, and the form data plus the procedurally-generated 2D barcode that IRCC's intake system machine-reads both live

inside the XML, not in the PDF body. When you try to process one through Merge or any other tool that mutates the PDF stream, the result is a blank page or a stripped barcode, and your client's filing is materially broken.

Every tool that ingests a PDF runs an XFA detection step alongside the encryption pre-flight. The detection is three-layered: a fast byte-scan of the first 256 KB looking for the NeedsRendering flag and AcroForm dictionary, a definitive pdf.js metadata check for IsXFAPresent (the authoritative flag the worker-side metadata returns), and a worker-side reactive backstop when pdf-lib refuses a file with a non-XFA error code that turns out on closer inspection to be XFA. When the detector trips, the file ingest paints an amber warning row with a single instructive paragraph: 'This is an IRCC dynamic form (XFA). Open it in Adobe Reader, use File ’ Print ’ Save as PDF to flatten it, and re-upload the flattened copy.' The tool's process button stays disabled until the flagged file is removed.

> **Important**: Adobe Reader's native Print ’ Save as PDF (or the Acrobat Pro equivalent) is the only path we recommend. It preserves both the filled field data AND the 2D barcode IRCC reads at intake, baking them into a regular flat PDF that every subsequent tool handles cleanly. pdf.js's experimental XFA renderer can produce visually-correct flattened pages but does not reliably emit the 2D barcode, so a tool that 'converted' an XFA in-browser would silently produce filings that fail IRCC's machine-read step. We chose to warn-and-block rather than ship a feature that produces unusable output.

Save to Drive

Every tool's output card carries a Save to Drive button alongside Download. Clicking sends the OUTPUT file (never the original) directly to your connected Google Drive or Microsoft 365 OneDrive folder under a fixed path: RCIC App / Secure PDF Tools / YYYY-MM-DD. The date subfolder uses UTC for consistency, so a Toronto-evening save lands in 'today UTC' (which might already be 'tomorrow' on the wall clock) — practical implication is that a long working session may span two date folders if you happen to cross UTC midnight, which is rare in normal

usage. The Save button is hidden when no Drive is connected; connect once from Settings ’ Drive Settings and every Secure PDF Tool picks it up immediately.

Architecturally, Save to Drive is the one place where the output file briefly transits our infrastructure. The Vercel function that talks to Google's and Microsoft's APIs has a 4.5 MB request-body cap; a typical compressed multi-page PDF is well under that, but an OCR-overlay export or a high-resolution Convert-to-PDF output can exceed it easily. The workaround is the signed-URL pattern: your browser asks the server to mint a signed upload URL into a tenant-scoped storage path, your browser PUTs the bytes directly to that URL, then the server downloads from storage and forwards to your Drive. The temporary storage object is deleted immediately on success; a daily cron sweeps any orphans one hour after creation. Bytes are at rest in our storage for typically a few seconds — long enough for the function to fetch and forward.

> **Note**: Save to Drive is the one privacy carve-out from the 'files never leave your device' promise. If you need the output to stay completely off our infrastructure, use Download instead — the file goes straight from the Web Worker to your local disk through the standard browser download path, no server hop. The trust badge SHA-256 hash on the output card lets you verify that the downloaded file matches the bytes the tool produced (and the same hash that the Drive-saved copy carries).

Email this file

Every Secure PDF tool's output card carries an **Email this file** button alongside Download and Save to Drive. Click it to send the output PDF directly to up to 3 recipients in each of the To, CC, and BCC fields (so up to 9 mailboxes per send), with a subject and an optional message. The flow is the same on Compress, Split, Merge, Convert to PDF, PDF to Images, OCR, Organize, Protect, Stamp, Redact, and Sign — every output you can download, you can also email.

When you click Email this file, a modal opens with one row per recipient field, a subject line pre-filled with a friendly default, an editable attachment filename (defaults to whatever the Download button would have used), and a small message box. On submit, the output bytes are uploaded directly to a temporary tenant-scoped Supabase storage path via a one-shot signed URL, the server downloads them, sends the email as an SMTP attachment under your firm's branded sender (**<Your Firm> via RCIC App**), sets your firm's email as Reply-To so any reply lands with you directly, then deletes the temporary storage object. A daily cron sweeps any orphans within the hour.

The attachment cap is 25 MB. That is the universally-enforced inbound limit across Gmail, Outlook, and most mailbox providers — sending a 30 MB attachment usually means the recipient never sees it. Files larger than 25 MB get refused in the modal with a friendly steer to use Save to Drive instead. If you put the same address in more than one slot (e.g. To and CC), the recipient gets exactly one copy on the highest-priority slot; this matches Gmail, Outlook, and Apple Mail behaviour.

> **Privacy note.** Email is not encrypted end-to-end. For sensitive client documents — identity scans, refusal letters, biometrics — prefer Save to Drive (the file lands in a permission-scoped folder you control) or share through Transfer Room (purpose-built encrypted client exchange with auditable handoff). The Email this file action exists for routine sends where convenience is the main concern, not for confidential dispatches.

Multi-file batches and Zip-all

Most tools accept multiple files in one session (up to twenty per session, with per-file memory limits enforced). The queue is sequential: the Web Worker processes one file at a time, posts progress messages back to the page, and moves to the next when the current one completes or fails. Each file's result card carries Download and Save-to-Drive buttons independently, so you can grab the early-finishing ones while later ones are still running. When every file is done, a Download all (.zip) button appears that bundles every successful output into a single ZIP via JSZip (dynamically imported only when needed — the library does not weigh down the initial page load).

Per-file errors are isolated: when one file fails (encryption pre-flight, XFA detection, out-of-memory on a very large input, or any other engine-level error), the row shows the localized error message and the queue continues with the next file. You can re-add a fixed copy of the failed file to the same session without restarting; the queue picks it up and processes it on the next available worker slot. Save-to-Drive on bulk is also available: a Save all to Drive button next to Download all uploads every successful output to the same date folder, with a real progress bar and a working Stop button that aborts the loop cleanly (the file in flight finishes, the queue does not start the next one).

Output validation and the trust badge

Every tool's output card shows a SHA-256 trust badge: a sixteen-character hash preview with a Show full toggle that expands to the full 64-character hex string. The hash is computed in the Web Worker over the final output bytes via SubtleCrypto.digest. Its purpose is verification: when you Save to Drive AND Download the same output, the trust badges on both copies are identical, so you can confirm the cloud copy and the local copy are byte-for-byte the same. The badge is output-only by design — we deliberately do not hash the input, because the input is your file and you already have it; hashing the input would add no information while the output hash adds verification of what the tool actually produced.

The Compress tool additionally runs a structural validation pass on its output before finalizing. The pass re-parses the compressed PDF through pdf.js and calls getOperatorList on every page; if any page fails to parse or render, the validation gate reverts to whole-file fallback — the original input bytes are returned instead, with a clear status note. The fallback exists because Compress is the only tool that mutates image XObjects in place, and the validation pass guarantees structural integrity. The pass adds a few seconds to the end of a long compress job; while it runs, the progress label flips to 'Verifying output...' so you know what is happening.

Compress

Compress shrinks PDFs by recompressing image XObjects in place (the same image objects pdf-lib sees in the document tree), without touching text, vector graphics, fonts, content streams, page structure, annotations, or any other non-image content. The implication: text stays selectable and searchable, vectors stay sharp, and the document's logical structure is byte-for-byte unchanged except for the swapped-in JPEG bytes on image streams. A separate lossless Step 9 pass at the end re-deflates every non-image FlateDecode stream at zlib level 9 (the legal maximum) so non-image bytes also shrink without any quality compromise.

Three compression tiers

- Maximum: aggressive image downsampling (720-pixel long-edge cap) plus JPEG quality 30. Aims at roughly Ghostscript /screen output — small file, screen-readable quality, suitable for emailing scanned bundles where the recipient just needs to read them.
- Balanced (default): moderate image downsampling (1100-pixel long-edge cap) plus JPEG quality 45. Aims at roughly Ghostscript /ebook output (150 DPI displayed) — the tenants-feel-good default that produces visibly clean documents at a meaningful size reduction.
- Quality-first: light image downsampling (1800-pixel long-edge cap) plus JPEG quality 70. Between Ghostscript /ebook and /printer — minimal visible quality loss, suitable for documents that need to remain print-quality after compression.

Two optional add-ons sit under the tier picker. Strip metadata removes the document's Title, Author, Subject, Keywords, Producer, and Creator fields plus the embedded XMP metadata block — useful when you want to share a PDF without revealing which tool produced it or who the author was. Convert images to grayscale runs a BT.601 luminance pass on every recompressed image before the JPEG encoder sees it; mozjpeg's 4:2:0 chroma subsampling then encodes the near-constant Cb/Cr channels almost for free, yielding roughly twenty to thirty percent additional savings on photo-heavy bundles (passport scans, signed documents, image-bearing IRCC letters). Both options are off by default.

Split

Split breaks one PDF into multiple output PDFs. It carries three operating modes accessible as tabs at the top of the workspace. Custom Ranges: type page ranges in the IRCC-familiar syntax (1-5, 8, 12-15, 20-) and the tool produces one output PDF per range, named with the source filename plus the range suffix. The trailing-hyphen form (20-) means 'from page 20 to the last page'. Every N Pages: pick an interval and the tool produces ceil(totalPages / N) outputs, each containing N consecutive pages. Visual Picker: see thumbnails of every page in a scrollable grid and either click to set cut-points (each cut starts a new output) or click to select individual pages for extraction into one combined output.

Thumbnails in Visual Picker render lazily through IntersectionObserver: only the visible-on-screen tiles fire their pdf.js render call, batched in groups of ten as you scroll. The first twenty render immediately so the grid never looks empty. Beyond two hundred pages the tool caps the picker and falls back to Custom Ranges (the visual UX stops being useful past that count). Page rotation badges are visible on every tile and carry through to the output; the tool preserves the source rotation on every kept page so a portrait page does not flip to landscape in the split output.

Merge

Merge combines multiple PDFs and images into one output PDF. It carries two modes accessible from the workspace toggle. File Mode (default): drop multiple PDFs and images (JPG, PNG) into the workspace, drag the file cards to reorder, name the output, click Merge. Each PDF contributes all of its pages in order; each image contributes one page. Page Mode: switch on the Show pages toggle to expand every source into a flat thumbnail grid where you can interleave pages from different sources, drag any page to any position, rotate individual pages (90 degrees clockwise per click), duplicate pages, exclude pages without removing the source, and delete pages permanently from the merge plan. The output PDF respects the exact page order you assembled in the grid.

The 300-page cap on Page Mode is a memory hint rather than a hard wall — the IntersectionObserver lazy-thumbnail pattern keeps a 300-page grid responsive, but past that the tab's memory footprint grows uncomfortably on typical laptops. The Add more files button accepts additional sources mid-session without restarting; new pages append to the end of the plan, then you drag them where you want them. Slot rotation badges show the current rotation on each tile; the rotation applies to the output page only (the underlying source PDF is unchanged on disk, which is consistent with the no-bytes-leave-the-tab promise).

Convert to PDF

Convert to PDF turns images and Word documents into PDFs. Accepted input formats: JPG, PNG, TIFF (multi-page), HEIC and HEIF (iPhone photos), and DOCX (modern Word documents). One settings panel governs every conversion: page size (Auto follows the source aspect ratio, Letter is 8.5 x 11 inches, A4 is 210 x 297 millimetres), fit (Fit shrinks to fit the page with letterbox margins, Fill scales to cover and crops the excess, Actual uses the source pixel size at 72 DPI), margins (None, Small, Standard), quality (Best, Balanced, Smaller — controls the internal JPEG re-encode of bitmap inputs), and output mode (One PDF combines every input into one multi-page document, Separate per file produces one output PDF per input file).

Architecturally, JPG and PNG go through the Web Worker via createImageBitmap and OffscreenCanvas; TIFF goes through the Worker too via utif2 (multi-page TIFFs emit one PDF page per TIFF page); HEIC and HEIF run on the main thread via heic-to (a libheif wrapper) and produce JPEG bytes that the Worker then assembles into a PDF page; DOCX also runs on the main thread via docx-preview (it needs a DOM to render Word styling) plus html-to-image to capture each rendered Word-page as a PDF page. The split between Worker-side and main-thread engines is invisible to you — the orchestrator routes each input file to the right engine and merges the per-source PDFs at the end when you picked One PDF output mode.

> **Important**: DOCX pagination is approximate. docx-preview lacks Word's exact layout engine; we honour Word's saved page-break hints, which produces a close-but-not-pixel-perfect page count compared to

> opening the document in Word itself. A blue review-before-use notice appears on the Convert workspace when any DOCX input is in the queue. The legacy .doc binary format (Word 97-2003) is not supported by docx-preview and is rejected at ingest with a message suggesting Save As .docx in Word first. HEIC image sequences (rare; iPhone Live Photos sometimes wrap multiple frames) emit the first frame only.

PDF to Images

PDF to Images is the reverse of Convert to PDF: drop a PDF, pick an output format (JPG or PNG) and a DPI (75, 150, or 300), and the tool produces one image per page. The output card lists every page-image with its own Download button, plus a Download all (.zip) button that bundles the entire set. Useful when you need to attach individual pages to an email (some recipients prefer images to a multi-page PDF attachment), when you want to embed a single page as a figure in a client memo, or when you need to drop a particular page into a tool that accepts images but not PDFs.

DPI matters for downstream use. 75 DPI produces small files suitable for screen-only viewing (a typical letter-sized page renders to roughly 800 KB per image at JPEG quality 80). 150 DPI matches IRCC's typical scan density and is the right setting when you need an image to look like a scanned document. 300 DPI matches print quality and the corresponding file sizes (each page image runs to several megabytes at JPEG quality 80; PNG is larger again). PNG is recommended when the page contains line art, screenshots, or text that needs to stay crisp; JPG is recommended when the page is a photograph or photo-heavy scan.

OCR

OCR (Optical Character Recognition) makes scanned PDFs searchable and selectable. The input is a PDF whose pages are bitmap scans rather than rendered text; the output is a new PDF that looks visually identical but carries an invisible text layer overlaid behind every word so you can highlight, copy, paste, and Ctrl-F-search the document content. The engine is Tesseract.js (tesseract.js-data EN language pack baked into the Worker bundle). Every

page rasterizes at 200 DPI through pdf.js, Tesseract runs the recognition pass, and the recognized words are placed into the output PDF as positioned invisible-glyph runs through pdf-lib.

OCR is computationally expensive. Expect roughly two to five seconds per page on a modern laptop, longer on lower-end machines. The progress card shows the current page number and the running total elapsed time. The English-only language pack is bundled to keep first-load fast; if a document is heavily multilingual, the recognition quality drops on the non-English content (Tesseract still produces output but may transcribe poorly on non-Latin scripts). The output text layer is best-effort and not a substitute for a human review — small or stylized fonts, low-resolution scans, and handwriting may all transcribe imperfectly. The visible page image is always preserved exactly so the legal document is unchanged.

Organize Pages

Organize Pages is the canvas-style page editor. Drop a PDF and the workspace shows every page as a draggable thumbnail tile in a responsive grid. Drag any tile to a new position to reorder. Click the rotate icon on a tile to rotate 90 degrees clockwise (repeated clicks rotate further; rotation persists into the output). Click the delete icon to remove a page from the plan (the source PDF is unchanged; the output PDF simply omits that page). Click the duplicate icon to insert an exact copy of the tile right after itself. Click Insert blank page to add a blank page at any position. Click Insert another PDF to drop a second PDF whose pages all append to the end (you then drag them where you want them).

Insert blank page lets you pick the page size (Letter, A4, or Match neighbour — the latter copies the size of the page immediately before the insertion point so the document size stays consistent). The blank page is exactly blank — no header, no footer, no page number; if you need any of those, use Stamp on the output. The Apply button at the top right runs the plan: the Worker assembles a single output PDF that reflects every rotation, deletion, duplication, and insertion in the exact order you arranged in the grid. The output preserves text, vectors,

fonts, and annotations on every kept page — Organize never rasterizes anything, so the document's text-selection and search continue to work in the output exactly as they did in the input.

Protect

Protect adds a password to a PDF. The password type is your choice: a user password requires the password to open the document at all, an owner password allows opening for reading but blocks editing, printing, or copying without the password (controlled by the permissions you tick), or both. The encryption runs in the Web Worker via the @cantoo/pdf-lib encryption API (AES-256, the modern PDF standard); the encrypted output is downloadable or save-to-Drive. The original file is unchanged. Protect carries its own pre-flight that detects an already-protected input and rejects it with a clear 'already password-protected' message — the resolution is to remove the existing password in your PDF reader of choice before re-uploading.

Permissions control what someone holding the user password (but not the owner password) can do with the document. The tickboxes are: allow printing, allow modifying, allow copying text and images, allow annotating, allow filling forms, allow assembling pages, allow accessibility tools (screen readers). The Adobe convention is to allow printing and copying and accessibility but block modifying and annotating and filling-forms and assembly — leaving the document readable and usable but tamper-evident. The tool's defaults follow that convention. You override them per-document. Picking the password itself: the workspace shows a generated-strong-password suggestion that you can replace with your own; the suggestion is twenty-character mixed alphanumeric with no ambiguous characters (no 0/O, no 1/l/I).

> **Important**: The password is never sent anywhere. We do not record it, we cannot recover it for you, and we cannot decrypt the output if you lose it. Save the password somewhere safe (your firm's password manager is the right place) immediately after creating the protected PDF — there is no support path for password recovery on an output we never had access to.

Stamp

Stamp adds visible marks across every page of a PDF. Three independent stamp types are available and any combination can be applied in a single pass. Watermark: large semi-transparent text rendered diagonally across the page center (common uses: DRAFT, CONFIDENTIAL, COPY, your firm's name on review documents). You pick the text, the font size, the rotation angle, the colour, and the opacity. Header and footer: standard header and footer text strings on every page, with separate left, centre, and right slots in each. Page numbers: typeset numbers (e.g. 'Page 1 of 12') in a corner of your choice — top-left, top-right, top-centre, bottom-left, bottom-right, bottom-centre. You pick the format string and the starting number.

Each stamp type carries its own per-page rendering pass. The Worker walks every page in the input PDF, draws the requested stamps on top of the existing page content, and writes the modified page back to the output PDF. The original text, vectors, fonts, and annotations are untouched — the stamps are an overlay layer, not a re-render. The diagonal watermark uses pdf-lib's drawText with a rotation angle; the rotation pivot is the baseline-left of the first glyph, so we compute an offset to land the rotated visual centre of the text on the visual centre of the page (this matters because pdf-lib's default rotation would otherwise pivot around the baseline-left and offset the text off-centre). Page numbers and headers/footers are typeset with the same Helvetica family the rest of the platform uses for chrome text.

Redact

Redact removes sensitive content from a PDF. Two modes: text search and area selection. Text search: type one or more search terms, the tool finds every occurrence across every page (case-insensitive by default), and you tick the occurrences you want to redact. Area selection: switch to a manual mode, see each page as a thumbnail, click and drag boxes over the areas you want to redact. Both modes converge on the same Apply step: every targeted page is rasterized at 200 DPI, the redaction rectangles are drawn as solid black fills on top of the rasterized

image, and the page is replaced in the output PDF with the redacted image. The non-redacted pages are passed through unchanged.

Rasterizing the targeted page is the safety property. A common mistake on weaker redaction tools is to overlay a black rectangle on top of selectable text — the text underneath remains in the PDF stream and a downstream user can select-and-copy through the black rectangle to recover the redacted content. Our Redact never leaves the original text in place on a targeted page: the page becomes a bitmap image with the black rectangle baked in, so there is nothing for a downstream user to select. The tool verifies the rasterization after every apply: it re-parses the output and confirms that every targeted page is image-only with no text operators. If the verification fails for any page, the output is held back with a clear error rather than shipped (the verify-fail path has never fired in production but it exists as a defence in depth).

> **Important**: Redaction is irreversible by design. The redacted output PDF has no recoverable record of what was underneath the black rectangle — the bytes are gone. If you redact the wrong area or the wrong text, re-do the redaction from the original PDF (which you still have on your disk) rather than trying to recover from the redacted output. Test your redaction plan on a copy before applying to the canonical version of the document.

Sign PDFs

Sign PDFs is the eleventh sub-tool and the only one that runs server-side. The use case is DocuSign-style electronic signing: you upload a PDF, place signature and initial fields where your signers should sign, name your signers with their email addresses and signing order, send, and the platform handles the rest. Recipients receive a tokened portal link, gate the link with an email confirmation plus an optional five-digit access code, sign through the portal (typed, drawn, or uploaded image), and the final executed PDF is assembled with an audit certificate appended and emailed to everyone who participated. The tool sits inside Secure PDF Tools because the source

format and the output format are both PDFs and the operational fit is the same; the architectural break from the other ten tools is documented up front on the upload form.

Recipient roles

- Signer: the recipient sees the document, fills any text fields you placed, applies their signature on the signature fields you placed, applies their initials on the initial fields you placed, and submits. Up to ten signers per envelope.
- CC participant: receives a courtesy copy of the executed PDF after every signer has signed, with no input requested. Up to five CC participants per envelope. Useful for the accountant, the senior counsel, the client's lawyer, or anyone else who should hold a copy without being a party to the signing.
- Preparer: that's you, the person who uploaded the document and configured the envelope. The preparer always receives the executed PDF on completion (deduped against signers and CC participants — if you are also a signer or already on the CC list, you receive one copy, not two or three).

Signing modes

Pick one of two signing modes on the configure step. Parallel: every signer receives the invitation at the same time and can sign in any order; the executed PDF assembles as soon as the last signer signs. Sequential: signers receive their invitation in order; signer two does not see the document until signer one has signed, and so on. Pick parallel when speed matters and the signers are independent; pick sequential when the document carries a who-signs-first ordering convention (typically the client signs first, then the consultant counter-signs). Auto-reminders fire daily after a configurable delay (default 7 days, up to 30 days) up to a configurable maximum (default 3 reminders, up to 5).

Field placement

Eight field types are available: signature, initials, full name, date signed, email, custom text, checkbox, and dropdown. You click-to-place each field on the document preview, drag to reposition, resize from any corner,

and assign it to one of your signers (each field is owned by exactly one signer — the others cannot interact with it). Signature and initials accept three input methods at signing time: typed (the signer types their name and the platform renders it in a script-style font), drawn (the signer draws with a mouse or touch on a signature canvas), or uploaded image (the signer uploads a transparent PNG of their handwritten signature). Custom text fields can be pre-filled with a default value that the signer can edit, or marked read-only. Date signed auto-fills with the signing timestamp in the signer's locale.

Identity gating

Recipients land on /pdf-sign/<token> from their invitation email. The page first asks them to confirm their email address matches the email on file (a soft anti-forwarding gate; the URL token is the actual capability). You can optionally require a 5-digit access code in addition — a code you set per-recipient when configuring the envelope and communicate out of band (a phone call, a separate email). Five wrong access-code attempts lock the recipient's gate for ten minutes; a fresh access code does not need to be issued — they just wait out the lockout and try again. The portal session lasts thirty idle minutes; sign-out clears the session cookie; the URL token still resolves so the recipient can come back later and re-gate.

Completion and the audit certificate

When the last signer signs, the platform assembles the executed PDF: every signer's signature, initial, and field input is rendered onto the original document at the positions you placed them; an audit certificate page is appended at the end of the document with the envelope reference, every signer's name and email and IP (redacted to /24), every signing timestamp in UTC, and a hash of the executed document; the merged document is encrypted with an owner password (random per envelope, discarded after; permissions allow printing and accessibility, block modify and annotate and form-fill and assembly) so the final PDF is tamper-evident. The executed PDF is attached to a completion email for every signer, CC participant, and the preparer (deduped). The PDF is also saveable to your connected Drive folder under RCIC App / Secure PDF Tools / Signed PDF files / YYYY-MM-DD.

Pre-filled saved signature for Premium-tier RCICs

When a signer opens a Sign PDFs link and three conditions all hold — the signer's verified email matches an RCIC member of the envelope's OWN tenant, that member has saved a signature in Profile and Signing and Signatures, AND the tenant is currently Premium-active — the signing card pre-fills the saved signature automatically. The signer sees an emerald banner that reads **Using your saved signature** with the rendered image, and every signature field they click stamps with that image. A **Sign fresh instead** button on the banner switches them back to the normal draw, type, or upload flow at any time; once switched, a maroon **Use my saved signature** button reappears so the signer can restore the pre-fill with one click.

The privacy scope is intentionally tight. The lookup only ever queries the envelope's OWN tenant. If the signer is also a Premium-tier RCIC of a DIFFERENT tenant (for example, they belong to two firms), the other tenant's saved signature is never surfaced into this signing — a saved signature is the licensee's firm-bound artifact, and pulling it across a firm boundary would leak it into a relationship the signer never opted into for this signing. The cross-tenant invariant is locked by a unit test that asserts even when the signer's email matches a member of Tenant B, only Tenant A's company id ever enters a scoped query.

Who benefits from this in practice: multi-RCIC firms doing internal co-signs (RCIC A sends a signing envelope to RCIC B; RCIC B opens the link, the saved signature pre-fills, one click stamps every field), self-tests by the firm Owner (you open your own envelope to verify it's set up right and the saved signature applies without re-typing), and any signing where the named recipient happens to also be a Premium-tier member of the envelope's tenant. External recipients — clients, opposing counsel, third-party payers — correctly never see this pre-fill; their flow stays exactly the normal draw / type / upload they have always had.

> **Setup.** Saved signatures are configured at Profile ’ Signing & Signatures. Pick a method (type with a cursive font, draw with the trackpad, or upload a PNG of your inked signature on a scanned page), preview, and save. The image is encrypted at rest under your tenant's data-encryption key (the same key envelope every other

> PII column uses). The pre-fill flow decrypts on demand at signing time and never exposes the bytes outside the signer's authenticated portal session.

Why Sign PDFs is server-side

Signing requires coordinating actions that happen days or weeks apart in different browser sessions on different devices. A client might sign on her phone at 11 PM on Sunday; the consultant counter-signs from her office laptop on Tuesday afternoon; the auto-reminder cron fires on Friday morning to nudge a sluggish signer; the executed PDF assembles on Saturday at 3 AM when the last signer finally signs. None of that can live in a browser tab. The Sign tool runs server-side so the envelope, the document, the per-recipient tokens, the audit ledger, the field placements, and the reminder cron all persist beyond any single session.

The encryption posture matches the rest of the platform's encrypted modules (Service Agreements, Co-Counselling Agreements, Active File Review, Transfer Room). The uploaded document is encrypted at rest under your tenant data-encryption key (DEK), wrapped under the master key-encryption key (KEK) that lives in Vercel environment variables, not in our database. Decryption happens server-side on a per-request basis when an authorized actor (you, an invited signer with a valid portal session, the assemble pipeline at completion) needs to see a value; the decrypted bytes are discarded immediately after the response completes. Even an attacker with full database and storage access cannot read the document without also obtaining the master KEK. The executed PDF in the cloud storage bucket follows the same posture, encrypted under the same DEK; downloading through the dashboard decrypts on the way out.

The audit ledger is the canonical evidence trail for any future legal challenge to the executed PDF. Every state change writes an INSERT-only row: envelope created, recipients added, sent, recipient gated, recipient signed, recipient declined, reminder sent, completed, downloaded, deleted. Each row carries timestamp, actor (your user id or the recipient's identifier), IP redacted to /24, and a JSON payload with the structured details. A Postgres trigger raises on any UPDATE attempt so the ledger is tamper-evident even from the service role. The audit

certificate page appended to the executed PDF summarizes this ledger in human-readable form so the document and its audit travel together.

Open-source engines and licences

Every PDF processing engine in this module is open source under a permissive licence — Apache-2.0, MIT, BSD-2/3, or zlib. We deliberately avoid AGPL and GPL engines (the AGPL is a poor fit for a hosted web product; the GPL conflicts with the closed-source nature of our broader platform), which is why Ghostscript is not in the engine list even though it would otherwise be the obvious choice for some operations. The engines are loaded lazily where possible to keep first-page-load fast and to avoid pulling WASM bundles you do not need.

- pdfjs-dist (Apache-2.0): the parsing and rendering engine across every tool. PDF text extraction, page rasterization, structural inspection, encryption pre-flight detection.
- @cantoo/pdf-lib (MIT): the PDF mutation engine. Page reorder, page rotation, page insert, image XObject replacement, encryption add (Protect), AES-256 protection at finalization (Sign).
- @jsquash/jpeg (MIT): the mozjpeg encoder. Image recompression in Compress, Convert to PDF, and the redaction-rasterization step in Redact.
- pako (MIT AND zlib): the lossless deflate engine. Step 9 non-image stream re-deflate in Compress.
- tesseract.js (Apache-2.0): the OCR engine in OCR.
- utif2 (MIT): the multi-page TIFF decoder in Convert to PDF.
- docx-preview (Apache-2.0): the DOCX renderer in Convert to PDF.
- html-to-image (MIT): the HTML-to-canvas capture step that bridges docx-preview's rendered Word pages into PDF pages.
-

heic-to (LGPL-3.0 — the one carve-out): the libheif wrapper that decodes HEIC and HEIF iPhone photos in Convert to PDF. LGPL is acceptable because we satisfy the relink requirement: heic-to dynamically loads its WASM at runtime rather than baking it into our app bundle, so a recipient who wants to use a different libheif build can swap it in.

- JSZip (MIT): the multi-file bundling engine. Download-all-as-ZIP across every multi-file tool.
- @napi-rs/canvas (MIT, server-side only): the Skia-backed canvas binding used by Sign for server-side PDF rasterization.

The full third-party licence text is at docs/secure-pdf-tools/THIRD-PARTY-LICENSES.md in the source tree and is accessible to any tenant on request. The single LGPL component (heic-to) carries the four compliance obligations enumerated in that document: a notice (the doc itself), source availability (links to the upstream libheif repository), replaceability (the WASM is dynamically loaded, not statically inlined into our app bundle), and a convey-on-request mailing address (info@investatech.com).

Integrations with other modules

Secure PDF Tools is intentionally a stand-alone utility module — none of the eleven tools depends on a record from another module, none writes back to another module, and a tenant who uses nothing else on the platform can still get value from the tool set. That said, the operational fit with the rest of the platform is intentional: when you compress an SA signed PDF before sharing it via Transfer Room, when you split a multi-applicant intake bundle before triaging each applicant separately in Active File Review, when you OCR a scanned refusal letter before quoting from it in a Co-Counselling Agreement's matter description, or when you sign a privacy-acknowledgement document outside the SA module's own signing flow, the tools fit naturally into your daily practice.

-

Transfer Room: many tenants run Compress on a fully-signed SA before uploading it to a Transfer Room transfer so the client downloads a smaller file. Save-to-Drive on the compressed output drops it into the same Drive folder Transfer Room provisions for the matter.

- Active File Review: AFR's public intake page accepts up to 50 MB per file and 150 MB total. When a client tries to upload an IRCC disclosure bundle that exceeds that, Compress in Maximum tier typically brings it under the cap with no readability loss. You can also recommend Split to break a multi-applicant bundle into per-applicant files before re-submitting.
- Service Agreements: the SA module's own signing flow handles SAs (with audit certificate, owner-password encryption, and the multi-party portal). Sign PDFs is for anything outside the SA module — a one-off privacy waiver, a Use-of-Representative form for a non-SA matter, a co-counsel acknowledgement that should not go through the CCA module's two-RCIC flow.
- Intake Forms and Bookings: tenants often run Convert to PDF on a smartphone photo of a passport or supporting document before attaching it to an intake or a booking note. Convert to PDF + Compress is a common two-step that turns a 12 MB iPhone HEIC into a 200 KB letter-quality PDF.
- Fax: outbound fax through the Fax module requires a PDF. When the source document is a Word file, a photo, or a scan, Convert to PDF is the first step before Fax. When the PDF is large, Compress in Maximum tier shrinks the transmission time and the per-page fax cost.

The privacy posture extends across the integrations. When you Compress an SA's signed PDF and then upload the compressed copy to a Transfer Room transfer, the compressed bytes never reach our servers during compression (they live entirely in your browser); they only reach our servers when you click upload in the Transfer Room transfer modal. When you Save to Drive instead of Upload, even that brief transit through our infrastructure

is skipped. The eleven tools chain together cleanly because each tool's input is the previous tool's output and the chain stays on your device end to end (until you decide otherwise).

Edit PDF — annotation canvas + fillable-form filling

Edit PDF is the twelfth Secure PDF Tool — a full-featured PDF editor in your browser. Drop any PDF and you get two distinct workflows depending on what the file is.

For ordinary PDFs: you get an annotation canvas on every page. Add text (with Bold / Italic / Underline + a fourteen-family font picker + multi-line support), click any baked-in text on the page to replace it in place (Edit text tool), draw shapes (rectangles, ellipses, circles, horizontal / vertical lines, freehand sketches), highlight or whiteout regions, and paste images. Every annotation has resize and rotate handles you can drag in the canvas. The inspector pinned above the page lets you fine-tune any selected object — colour, opacity, font, dimensions, rotation — without leaving the document.

For fillable PDFs: Edit PDF auto-detects every form field on every page and lets you fill it directly. Text fields, checkboxes, radio buttons, and dropdowns all surface as clickable inputs layered exactly over their rect on the page. The Form fields card at the top reports how many fields the engine found and which page each lives on; a flatten-on-export toggle bakes your values in so they're permanent (not editable) for downstream readers.

Open Edit PDF from the Secure PDF Tools hub (sidebar) or directly at /dashboard/secure-pdf-tools/edit. Per-file size cap: 100 MB. Page cap: 200. The editor runs entirely in your browser — your PDF never leaves your device for the work itself.

Edit PDF for IRCC forms (hybrid + dynamic-XFA)

Real-world IRCC forms come in two shapes, and Edit PDF handles both — but differently.

Hybrid AcroForm IRCC forms (IMM 5476 Use of Representative, IMM 5645 Family Information): these have real fillable fields on top of an XFA shadow layer. The standard PDF library that other web editors use refuses

these files mid-extraction. Edit PDF uses a dual-engine extractor — pdf-lib first, with pdf.js as a fallback — so when one engine refuses, the other takes over. Field values save back through whichever engine read them. You fill, you export, you open in Adobe Reader, your values are there.

Dynamic XFA IRCC forms (IMM 0008, IMM 5257, IMM 5406, IMM 5562, IMM 5669, IMM 5708, IMM 5709, IMM 5710): these forms have zero usable AcroForm fields — they need Adobe Reader's native XFA engine to render their layout AND to generate the 2D barcode IRCC scans at intake. Edit PDF detects them at upload via three layers (byte-scan hint, pdf.js metadata, a curated registry of the ten target IRCC forms) and paints an amber Adobe Reader Route card explaining the path: open the file in Adobe Reader, fill it there, and the resulting PDF preserves both your filled fields AND the barcode. Browser-side rasterizing would lose one or both — we don't pretend otherwise.

The Upload Diagnostic card at the top of every editor session names exactly which path your file took (Editor success / Adobe Reader Route / parse failure / etc.) + the classifier flags + the field engine that won. When something goes wrong, the card explains exactly which step failed.

Edit PDF — privacy + export destinations

Edit PDF is the eleventh of the twelve Secure PDF Tools that run entirely in your browser. The PDF you drop never leaves your computer for the editing work itself. You can verify this in your browser's DevTools (F12) ’ Network tab: zero outbound requests during the editing session.

When you finish editing, the export card offers four destinations:

- **Download** — saves the edited PDF locally with one click.
- **Save to Drive** — if you've connected Google Drive or OneDrive in Document Storage, saves the output directly to your linked drive under **RCIC App / Secure PDF Tools / <today's date in your timezone>**. Only the FINISHED output transits our infrastructure briefly on its way to your drive; the original stays on your device.

- **Email this file** — opens a To / CC / BCC form, attaches the edited PDF, and sends through your tenant's reply-to address. Useful for sending a filled IMM 5476 back to a client for review.
- **SHA-256 trust badge** — every output displays a SHA-256 hash of the bytes we produced. Downstream recipients can verify the file you forwarded matches the file we generated.

Edit PDF — click-to-replace existing text

Edit text lets you change words that are already baked into the PDF, not just add new text on top. Pick the tool from the toolbar, click any word or phrase in the document, and the original text is covered by a background-matched whiteout while a fresh editable box appears in its place, pre-filled with the original words at roughly the original position, size, and colour.

How the whiteout works: Edit text samples the pixel colour just outside the original text box and fills a solid rectangle of that colour behind the replacement box. This means edits on white paper backgrounds are invisible seams; edits on shaded or coloured backgrounds (form field boxes, header ribbons, cell fills) blend into their surroundings rather than showing a bright white gap. If the sampled colour is wrong for your case, you can adjust or delete the whiteout overlay directly from the layer list — Edit text creates it as a normal Whiteout overlay you can move, resize, or recolour.

Font matching: when you click a text run, Edit text reads the original font name and style flags from the PDF (Bold, Italic, Bold Italic, or Regular) and picks the closest match from the shared fourteen-family font picker. If the original was, say, Calibri Bold, the replacement inherits Calibri Bold; if it was a font Edit PDF doesn't know, it falls back to the nearest built-in that shares the same shape (serif ’ Times, sans-serif ’ Helvetica, monospace ’ Courier). You can override the pick from the inspector at any time — the font, size, weight, italic, and colour of the replacement are all editable like any other text overlay.

Everything else about Edit text is identical to the other Edit PDF tools: the change is a normal overlay you can select, move, resize, rotate, undo, or delete. It exports through the same pipeline (subset + embed at export, no server call). You can mix Edit text edits with new-text overlays and shapes on the same page. The tool runs entirely in your browser — the PDF you edit never leaves your device.

Edit PDF — fourteen shared font families

Edit PDF offers fourteen font families for both the Text tool (new text overlays) and Edit text (click-to-replace). The picker in the toolbar shows exactly the same list in both places, so a replacement can visually match its neighbours.

Three PDF built-ins (always available, no download required): Helvetica, Times, and Courier. These are the three families every PDF viewer natively renders — the smallest possible output because the reader supplies the font.

Eleven embedded open-license replacements chosen to be metric-compatible with the typefaces real-world documents (Word, IRCC forms, legal templates) use — the letter widths match the target typefaces closely, so a replacement lands at nearly the right width and fits the original layout:

- **Arial** ’ Arimo (Apache License 2.0, Google Fonts)
- **Times New Roman** ’ Tinos (Apache License 2.0, Google Fonts)
- **Georgia** ’ Gelasio (Apache License 2.0, Google Fonts)
- **Cambria** ’ Caladea (SIL Open Font License, Google Fonts)
- **Calibri** ’ Carlito (SIL Open Font License, Google Fonts)
- **Verdana** ’ DejaVu Sans (Bitstream Vera / DejaVu license, redistributable)
- **Courier New** ’ Cousine (Apache License 2.0, Google Fonts)

- **Garamond** ’ EB Garamond (SIL Open Font License, Google Fonts)
- **Palatino** ’ PT Serif (SIL Open Font License, Google Fonts)
- **Franklin Gothic** ’ Libre Franklin (SIL Open Font License, Google Fonts)
- **Consolas** ’ JetBrains Mono (Apache License 2.0)

All fourteen families support Bold, Italic, and Bold Italic. The embedded fonts subset + embed automatically at export via @pdf-lib/fontkit — only the characters you actually used ship in the output PDF, so file sizes stay small. The subset is written as a proper TrueType stream so Adobe Reader (and every other PDF viewer) can extract it cleanly.

Graceful fallback: each embedded family carries a nearest-built-in fallback in its metadata (Arial ’ Helvetica, Times New Roman ’ Times, Cambria ’ Times, etc.). If a font file couldn't be downloaded during setup (rare — only a CDN issue would cause this), the picker hides that family and its selection falls back to the built-in, so text still renders. You can see which families installed on this deployment by opening /fonts/edit/manifest.json — the available array lists every embedded family that's ready to use.