

RemoteAccessApp Security White Paper

How RemoteAccessApp protects the PCs it reaches, the accounts that reach them and the data in between. Written for IT teams, managed service providers and security reviewers, including where the design has limits.

Version 1.1 · October 2, 2026 · Rust Games Australia Pty Ltd (ABN 64 683 623 065)

1. Executive summary

RemoteAccessApp is remote access and remote support software for Windows 10 and 11. An **Agent** runs on each PC you want to reach. You connect from the **Console** (a Windows app), from a web browser through our website, or from an AI assistant through a local MCP server. Our servers handle accounts, sign-in and the relay that joins the two ends.

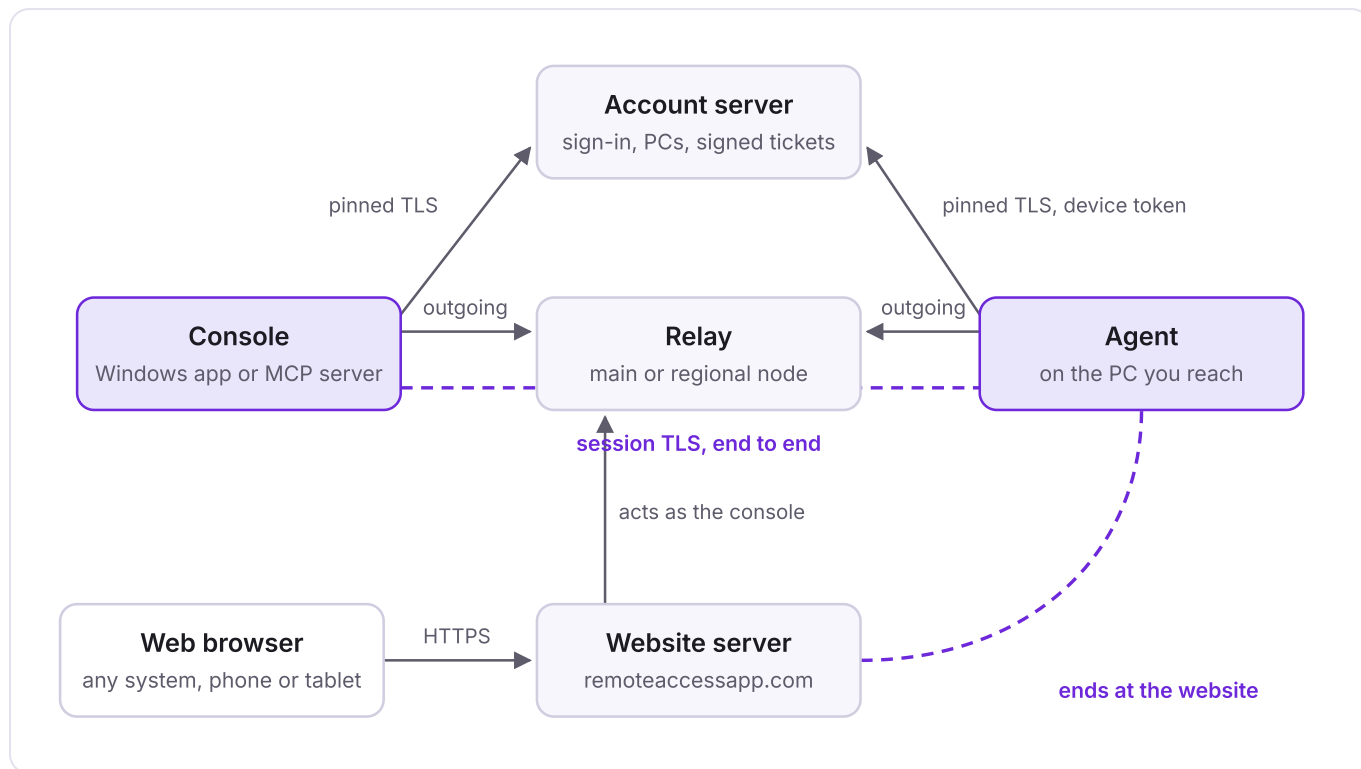
- **The person at the PC decides.** Unattended access is off by default. Until it's turned on at the PC, every session needs someone there to click Allow, and an unanswered request is denied after 45 seconds. A banner stays on the screen while anyone is connected.
- **Sessions from the app are encrypted end to end.** The Console and the PC run their own TLS session through the relay, which forwards bytes it doesn't decrypt. Neither side needs a port opened on a router or firewall.
- **The PC enforces what the server signs.** Ownership, two-step verification, plan and team permissions travel in short-lived tickets signed by the account server (ES256), which the Agent checks with keys built into it. A modified Console can't talk its way past them.
- **Your account password never reaches our server.** Current apps and the website send a key derived from it, and synced server passwords are encrypted end to end.
- **Updates are signed** with our own key and checked before they install, over connections pinned to our server's certificate.

There are limits too. Browser sessions are encrypted but not end to end, because our website acts as the console. The relay and the account server are part of the trust model.

Our installers aren't yet signed with a code signing certificate, and we hold no third-party certifications. Section 19 lists every limit we know about.

2. Architecture overview

PART	WHERE IT RUNS	WHAT IT DOES
Agent	Each PC you reach	Shares the screen and applies input. Runs as the signed-in Windows user, without administrator rights, and only connects out.
Console	Your Windows PC	Opens sessions; holds the SSH, SFTP, FTP, RDP and VNC clients. Its own TLS certificate identifies it to PCs. Also listens on TCP port 9991 for Agents set up to connect directly on a local network.
MCP server	Your Windows PC	Lets an AI assistant use your PCs as you, under the Console's rules (Pro).
Website	remoteaccessapp.com	The dashboard, plus remote desktop and SSH in a browser, where the website server is the console.
Account server	api. and updates.remoteaccessapp.com	Accounts, sign-in, PCs, teams, sync, alerts, PC management and billing records. Signs tickets and tasks; publishes signed updates.
Relay	The account server, plus optional regional nodes	Joins a console and a PC that both connected out to it, after checking the account may reach that PC.



Solid lines: TLS connections to our servers. Dashed: the session's own TLS between the console and the PC, which the relay forwards. In a browser session that TLS ends at the website server.

On a local network the Console can also accept Agents directly. It admits only Agents that prove the right key inside TLS, and locks out an address for 2 minutes after 5 failures in a minute.

3. Data flows

DATA	PATH	WHO CAN READ IT
Screen, input, chat, clipboard, sound, files, commands	Console or MCP server ⇌ relay ⇌ Agent, inside the session's TLS	The console and the PC. The relay forwards ciphertext.
The same, in a browser	Browser ⇌ CDN ⇌ website server ⇌ relay ⇌ Agent	Also the website server, in memory, and the CDN in transit (section 10).
Account password	Stays on your device	Nobody: the server gets a derived key.
Synced saved passwords and SSH keys (Pro)	Encrypted on your device first	Only someone with your account password.

DATA	PATH	WHO CAN READ IT
Saved hosts, scripts, settings	Synced to the account server	You and our server (encrypted at rest, not end to end). Never passwords or keys.
Session records	Written by the relay	Who, which PC, when, how long, how much data, how many files. No content or IP address. 90 days.
PC health (Pro alerts)	Agent ⇄ relay ⇄ account server	CPU, memory, free disk space, uptime, restart and update state. Latest report only, in memory.
Card details	Browser ⇄ Stripe	Stripe only.

The relay sees metadata: account and PC identifiers, network addresses, the console certificate's fingerprint, the account's email address, session times and byte counts. It doesn't see screens, keystrokes, files, clipboard or commands.

The relay is part of the trust model

The PC learns which console certificate to expect from the relay, and the signed ticket binds it. A relay that substituted its own certificate could sit in the middle of a session, but it couldn't produce a matching ticket: the PC would treat the link as an unknown console, with no unattended entry, and ask the person there, showing a Console ID that doesn't match the real one. In the other direction, a console relies on the account server to pair it with the right PC.

4. Encryption in transit and at rest

In transit

- **Console and PC:** TLS 1.2 or 1.3 with the operating system's cipher suites. The console's self-signed certificate (RSA 2048) is checked by its SHA-256 fingerprint, which over the relay must match the signed ticket for the link to count as the owner's or a team member's. On a local network the PC remembers the first console it meets and proves itself with a 160-bit key.

- **Apps and our servers:** TLS 1.2 or 1.3, pinned to the SHA-256 fingerprint of our server's certificate, which the apps carry. Any other certificate is refused, including one from a public authority. These endpoints aren't behind a CDN, and their front end allows only ECDHE key exchange with AES-GCM or ChaCha20.
- **Regional relay nodes** make their own keys. The apps trust a node only through key pins in a ticket the account server signed, and show it only short-lived tickets, never their sign-in or device tokens. Nodes authenticate to the account server with signed, time-limited requests.
- **Website:** HTTPS with HSTS; our server accepts only TLS 1.2 or 1.3. **Notifications:** Web Push messages are encrypted for your browser (RFC 8291), so the push service can't read them.

At rest

- On our server, authenticator secrets, synced documents, PC notes, push subscriptions and PC management parameters, scripts and results are encrypted with AES-256-GCM, using per-purpose keys derived with HKDF from a 32-byte server secret, each record bound to its account.
- Values we only need to check are one-way: password-derived keys (PBKDF2), session tokens and PC credentials (SHA-256), license keys, email links, sign-in and setup codes and invitations (HMAC). The account database is SQLite; the server secret and signing keys are files only the service can read.
- On your PCs, the Agent's credentials and trusted-console list and the Console's certificate and saved host passwords are protected with Windows DPAPI for the Windows user. The website server's own console certificate is sealed with AES-256-GCM under a key kept outside its data folder.

Limit

Email addresses, names, IP addresses and PC names are stored as they are, protected by the server's access controls. The server secret sits on the same server as the database and in its backups, so this encryption protects a leaked database file, not a compromised server. Backups are nightly, integrity-checked and kept on the server with owner-only permissions (the newest 14). The backup script encrypts them with age to a key the owner keeps off the server, and can copy only encrypted archives to off-site storage; until that key is installed on the server, they aren't separately encrypted.

5. Identity and authentication

- **Password never sent.** The apps (0.11.2 and later) and the website derive a key on your device with PBKDF2-HMAC-SHA256 (600,000 iterations, salted with your email address) and HKDF-SHA256, and send only that. The server stores a PBKDF2-SHA256 hash of it with a random salt, compares in constant time, and does the same work for unknown addresses. The password vault uses a different salt, so this key is no shortcut to it.
- **Two-step verification:** standard 6-digit authenticator codes, each usable once; 10 wrong codes in 15 minutes pause checking. The secret is encrypted at rest, and the 10 single-use backup codes are stored only as hashes. A password reset also needs a code.
- **Google and GitHub sign-in (optional):** the account server uses the providers' standard authorization-code flow, asking only who you are (openid, email and profile on Google; read:user and user:email on GitHub). Its one-time state is tied to the browser by an HttpOnly cookie and expires after 10 minutes. We keep the provider's account ID and email address, never its access token, and get no access to anything else in that account. An existing account is linked only when the provider has confirmed the email address, and two-step verification still asks for its code. The Windows and Linux apps sign in through your own browser and receive a one-time code on 127.0.0.1, bound to the app with PKCE.
- **Email sign-in codes:** without an authenticator, a new device or browser needs a 6-digit emailed code (10 minutes, 5 tries, stored as an HMAC, rate-limited per account and network). If email can't be sent, the sign-in is refused. The device can then be remembered for 30 days with a random token stored as a hash. Accounts with an unconfirmed address or created in the last day can't use the share of our email budget kept for established accounts, so mass sign-ups can't starve real users of codes and reset links.
- **Password guessing:** after 10 wrong passwords for one email from one network (a /24 or /48) in 15 minutes, that network waits out the rest of the 15 minutes before it can try that email again. Only the guessing network waits, so nobody can lock an owner out, and unknown addresses are paused the same way, so the answer says nothing about whether an account exists.

- **Sessions:** random 256-bit tokens stored as SHA-256 hashes, expiring 30 days after last use. On the website the token is in an encrypted `__Host-` cookie (Secure, HttpOnly, SameSite=Strict) that scripts can't read; changes also need an anti-forgery token, and open sessions recheck the sign-in every 15 seconds.
- **Revocation:** a password reset, by email link or license key, deletes every sign-in session, remembered device, notification subscription and pending code, replaces the account's long enrollment code and ends unused setup codes, ends live relay connections and tunnels, and sends a "password changed" email. Disabling an account ends its sessions, subscriptions and live connections too. Signing in on the website over an existing sign-in ends the old one on the server.
- **Recent sign-in** is required to allow shared server addresses in trusted networks (10 minutes) and to show the license key (password, plus an authenticator code unless you signed in within 10 minutes). An account made with Google or GitHub has no password to ask for, so turning two-step verification on or off, showing the license key, exporting data, deleting the account and changing trusted networks only work within 15 minutes of signing in.
- **Trusted networks:** limit your account and each PC to IP ranges you choose, enforced by the server at sign-in, on the API, at relay connections and for tunnels. Ranges wider than /8 (IPv4) or /32 (IPv6), and our shared server addresses unless you opt in, are refused.
- **Security emails** report password changes, two-step verification changes, trusted network changes and sign-ins from apps or networks we haven't seen before.

Limit

Apps older than 0.11.2 still send the password itself; the server then upgrades the account, and current apps never send it for that account again.

6. Consent and control at the PC

Asking first

Unless unattended access is on, the PC asks **Allow remote access?**, naming who wants to connect and the console's name and Console ID (a short form of its certificate

fingerprint), or a browser's name and Browser ID. Nothing from the PC is sent until someone clicks Allow. No answer in 45 seconds, closing the prompt or the console leaving all mean no. The prompt is hidden from screen capture, and remote mouse, keyboard and command-prompt input is ignored while it's open, so the person connecting can't answer it. Chat from a console that hasn't been accepted is held and shown only once the session starts (discarded on a refusal), and a sender name posing as RemoteAccessApp is replaced. The same console, signed in to the same account, may reconnect within 10 minutes without asking again; any refusal clears that.

Unattended access

Off by default. It's turned on at the PC, or a console asks and the person at the PC agrees there; turning it off from a console works at once. With it on, a connection gets in without a click only in these cases:

CONNECTION	RESULT
A console someone at the PC chose to trust	In without asking
A console or browser signed in to the account that owns the PC, with or without two-step verification, while "Let my own account in without asking" is on at the PC (the default)	In without asking (logged at the PC)
The same, but removed at the PC or with that setting off	Asked
A browser someone at the PC remembered, for the time they chose	In without asking, even with unattended access off
A team member, a local-network console, anything else	Asked, unless trusted at the PC

The trusted list lives only on the PC. The website's shared certificate is never trusted: each browser instead proves its own non-extractable ECDSA P-256 key (made with WebCrypto) by signing a one-time challenge bound to that PC and account. Removing a console or browser from the list ends its session at once.

Connection password

The person at a PC can also require a password before anyone connects: the apps, the website, an AI assistant and connections through the PC, the owner's own account included. It's set only at the PC (never while someone is connected) and is an extra step, not a replacement: the Allow prompt and the rules above still apply afterwards, unless they choose "Skip the Allow prompt after the correct password" (off by default). The password never leaves the device it's typed on: the PC keeps only a salted PBKDF2-SHA256 key (600,000 iterations), encrypted for that Windows user, and sends a one-time challenge bound to the PC and the connection's certificate; the client answers with an HMAC of it, so our relay and website only ever pass on a proof that can't be reused. Five wrong tries pause that console and the PC for 1, 5, 15, then 60 minutes, with a notice at the PC and, if alerts are on, a message to the owner. "Remember me on this device" (if the PC allows it, for up to 1, 7 or 30 days or until removed) keeps a token bound to that console's certificate or that browser's non-extractable key; changing the password ends every one, and the PC lists them for removal. Management tasks from the account aren't asked for it: they're signed by our server for that PC and follow the consent rules below.

During a session

- **A banner** stays on screen while a session or network forward is active. It can be collapsed, not dismissed, shows what's in use (screen, control, clipboard, sound, terminal, Windows tools) and names who is acting, including an AI assistant.
- **View-only at any time:** turning off "Allow remote control" stops input, the file browser and transfers, the command prompt, Windows tools changes, network forwards and management tasks that change the PC. The person can also quit RemoteAccessApp.
- **Activity:** notable actions appear as notices and go to a local activity log on the PC. Clipboard use is logged by type and size only. Anything typed at a password prompt is logged as "not shown", and likely secrets in command lines (password options, tokens, credentials in addresses) are masked.
- **Files** sent to the PC land only in a RemoteDesk folder under Downloads, with cleaned names, never overwriting anything. The file browser reaches whatever the Windows user can, and logs every action.
- **Commands and Windows tools** run as the Windows user without elevation; critical system processes and the Agent can't be ended.

- **Network forwards** (for browser SSH "through my PC") are offered only to the owner's account, always ask at the PC, and reach only port 22, never the PC itself or cloud metadata addresses.

Limit

The activity log's masking is a best effort based on common patterns: a secret typed into a command in an unusual form can still end up in the PC's log. Received files are marked as downloaded from the internet, so Windows and Office treat them with the usual caution.

7. Access control for teams

Teams (Pro) share chosen PCs and hosts. Members are Admins, Members or Viewers; the owner has every right. For each shared PC the owner or an Admin grants view, control, files, Windows tools and commands, and manage access, capped by role (a Viewer can only view). Enforcement happens in three places:

- **Server:** every team call checks the caller is the owner or a joined Admin. Admins manage only Members and Viewers, can't grant more than they hold, and nobody changes their own role. Sharing applies only while both sides have Pro. When a member is removed or their role or rights change, their open connections to the owner's PCs end at once.
- **Relay:** signs the member's effective rights into the ticket, and won't connect a limited member to an Agent too old to enforce them.
- **PC:** the Agent enforces those rights whatever the console asks: no input or clipboard without control, no files without files, no command prompt or Windows tools without tools, no access changes without manage. Team members are still asked at the PC unless trusted there.

8. Remote management safety

PC management runs scripts, restarts and Windows updates on many PCs at once (free), and with Pro winget app installs and jobs on a schedule. Because these reach a PC outside a session:

- **Tasks are signed.** Each job, and each PC's schedule set, is signed by the account server (ES256) and verified by the Agent, bound to one PC, the person who started it, their rights and the exact parameters, with an expiry (24 hours for a job). A job runs at most once; an older schedule set is ignored, and one not renewed for 30 days stops.
- **The relay only nudges.** The Agent fetches tasks itself over pinned HTTPS, so a compromised relay can delay a task but can't inject, change or replay one.
- **Consent applies.** A task skips the prompt only when unattended access is on and the person who started it owns the PC with two-step verification. Otherwise the PC shows the exact command and asks (5 minutes). Nothing that changes the PC runs while remote control is off. A team member needs the PC's tools right (and control for changes) for a job, and manage access as well for a schedule.
- **Restarts** follow a visible notice (normally 5 minutes), which can be postponed by an hour up to three times; if the notice can't be shown, the restart is cancelled.
- **Records:** every action shows a notice at the PC. Parameters, scripts (one-off and scheduled) and results are encrypted at rest; the command shown for confirmation is worked out from the encrypted parameters, not stored in the clear. Jobs and results are deleted after 30 days. The security log never holds script text or output.

9. AI assistants (MCP)

The MCP server (Pro) lets an assistant such as Claude Code list your PCs, take screenshots, click, type, run commands, use Windows tools and run commands on saved SSH servers. It runs on your PC and talks to the assistant over standard input and output; it opens no network port.

- It signs in as you and reaches PCs through the relay with its own certificate, so the same prompts, view-only switch, team rights and plan rules apply. The PC shows it as, for example, "Claude Code via RemoteAccessApp MCP" with the Windows user and PC it runs on.
- It can't transfer files, use the clipboard, hear sound or chat. Saved passwords are used without being shown to it.
- A new SSH server key is saved only if a fresh check shows the exact fingerprint you confirmed; a changed key is never replaced, and it can't add a different kind of key for a server that already has one (only the Console can).

- The Console's MCP page shows what the assistant is doing, with Disconnect and Stop buttons and an off switch.

Limit

Asking you before a step is up to the assistant's own tool approval, not something we can force. Under the own-account rule (section 6), an assistant signed in to the owner's account enters an unattended PC without a prompt, like the owner's Console. What it sees goes to its provider under their terms.

10. Sessions in a web browser

A browser can't hold the console's end of the session's TLS, so in a browser **the website server acts as the console**. It opens the TLS session to the PC through the relay with its own certificate, decrypts the picture and passes frames to your browser unchanged over a secure WebSocket, and validates every input message. For SSH in the browser, the website server sees the server password or key while it signs in.

- Frames, input and SSH credentials are handled in memory only, never recorded or logged.
- The PC never trusts the website's certificate for unattended entry; each browser has its own key, and the person at the PC is asked unless they remembered that browser or the owner rule applies.
- Sockets accept only same-origin requests from a valid sign-in, with limits per account and per visitor, and close when you sign out or reset your password.
- For SSH, a new server key must be trusted before the password is sent, and a changed key is refused. The website reaches only your saved hosts and refuses private, reserved and its own addresses.

What this means

A browser session is encrypted all the way but not end to end: our website server and the website's CDN (Cloudflare) are in the path. Someone in control of the website could watch or act in a browser session in progress, or use a remembered browser while it has the site open. When end-to-end encryption matters, use the Console.

11. Server and infrastructure security

- **Separation:** production runs on a single Linux server. Administrative SSH uses a separate address that does nothing else, accepts keys only, is open only to the owner's address and is watched by fail2ban. Public services bind explicitly to the service address, the firewall denies incoming connections by default, and security updates install automatically.
- **Least privilege:** the website, the account server and its front end run in containers as non-root users, with read-only root file systems, all Linux capabilities dropped, no new-privileges and resource limits. The account server sits on an internal network behind its front end.
- **Secrets** (payment, email and push keys) are separate root-owned files mounted read-only, not environment variables; other service tokens are in root-only files.
- **Front ends (nginx):** TLS 1.2 and 1.3 only, server version hidden, request-size limits. The account server's front end adds allowed methods per route, per-address rate and connection limits and strict headers.
- **Administration:** the admin interface listens only on the server itself and needs a long random token; the hosted admin dashboard adds an address allowlist, a passphrase and an authenticator code, with short sessions.
- **CDN:** only the website goes through Cloudflare. The account, relay and update endpoints don't, because the apps pin our certificate.

Limit

Production is one server in one location, a single point of failure for availability. Optional regional relay nodes (none run in production today) carry sessions but don't hold the account database. A node's tickets last at most 30 minutes and its revocations are replayed when it reconnects, but a compromised node is trusted like the relay for the sessions it carries and could misroute connections to PCs whose IDs it knows; it still can't read a session or skip the PC's consent.

12. Application security

- **Content Security Policy:** scripts only from our own site on every page, no inline scripts (the build fails on one) and no eval; no plugins, no framing, no base-URL changes. Inline styles are allowed.
- **Third-party scripts on public pages only:** the live chat (tawk.to) and Meta's ad-measurement pixel never run on the dashboard, remote desktop, the SSH terminal, checkout, or the emailed password-reset and email-confirmation pages. The server's per-page policy and the app both enforce this, and crossing between public and signed-in pages loads a fresh page. The pixel's automatic data collection is off and it respects Global Privacy Control. Stripe's script loads only on checkout. Cloudflare Turnstile (the sign-up form's bot check) is allowed only on public pages and checkout, and loads only with the sign-up form.
- **Headers:** HSTS, X-Frame-Options DENY, nosniff, Referrer-Policy no-referrer, a restrictive Permissions-Policy, same-origin opener and resource policies, and no caching of personal pages.
- **Abuse limits:** per-address rate limits on sign-in, sign-up, reset, two-step verification, enrollment, relay and tunnels; a per-account pause for a network that keeps guessing a password; per-account limits on emails and codes; request-size caps; Cloudflare Turnstile on sign-up (verified by our server, failing closed) and a self-hosted "I'm not a robot" check on the other open forms; session caps. The server tunnel (Pro) reaches only saved hosts and refuses private, loopback, link-local and cloud metadata addresses, checking the address it actually connects to.
- **Testing:** automated suites cover the server, website, shared libraries, Console and MCP server, including authentication, rate limits, ticket and update verification, TLS pinning, vault encryption, team permissions and signed tasks. End-to-end suites run the real stack locally: a server security suite (authorization, cross-account access, relay exhaustion, tunnels), a website security harness, browser journeys in headless Chrome, and Windows tests of the Console and Agent with real capture, TLS and relay.
- **Secret scanning:** a release-time scanner, run by hand, searches the built apps, the website and the source for our real production secrets and common key formats.

Limit

Releases don't yet run the test suites or the secret scan automatically; both are manual steps. Wrong passwords pause only the guessing network, never the account itself (so a guesser can't lock the owner out); a guesser spread over many networks is held back by the per-address limits and by the second step (an authenticator or emailed code). The chat and ad scripts on public pages, sign-in and sign-up included, run on the same web address as the signed-in app, so a compromised vendor script there could act for a visitor who is signed in.

13. Updates and software integrity

- **Signed manifests:** each release has a manifest signed with ECDSA P-256. The apps check it with a built-in public key before reading it, then check the installer's size and SHA-256. Manifests name their app and channel and expire, and the apps never install an older version. The signing key exists only on our release PC, under Windows DPAPI.
- **Pinned and patient:** updates download over pinned TLS. They are automatic and can't be skipped, but they never install during a session: the Agent waits until no one is connected, and the Console waits until its sessions end and then gives a 60-second notice. The website serves only installers it verified against a signed manifest.
- **Pushed, never trusted:** when we publish a release, our server tells online apps within seconds. That message only makes an app check early. The app still fetches and verifies the signed manifest itself, so a tampered relay or server can delay an update at most, never install one.
- **Update service:** on PCs installed for all users, a small Windows service running as LocalSystem applies updates without a permission prompt. It installs only a release signed with our key, newer than what's installed and for its own channel. It downloads into a folder only administrators can write, keeps the file locked from the hash check until it runs, and takes no file, address or command from other programs. Apps can only ask it to check now or to wait. It keeps a hash-chained log of every install and checks the installed files against their signed list afterwards.
- **Integrity monitor:** release builds carry a signed list of their program files with SHA-256 hashes and recheck themselves. A confirmed modified copy drops to the free

plan, reports itself at sign-in and offers Repair. Anything that matters for security is enforced by the server or signed tickets, not by this check.

In progress: code signing

Our installers and program files aren't Authenticode-signed yet, so Windows may warn the first time you run the installer. The build has a step ready for it. Until then, our signed manifests are what prove an update genuine.

14. Privacy and data retention

- Your email address is the only personal detail we need. We don't sell personal data, show ads or record sessions.
- The security log is deleted after 90 days, backups roll over within 14 days, and server request logs are overwritten after a short time.
- Service statistics contain no email, name, IP address or account identifier; unique visitors are counted with codes that change daily.
- Providers: Resend (email), Stripe (payments), Cloudflare (the website's CDN), your browser's push service (encrypted notifications), and on public pages only, tawk.to (chat) and Meta (ad measurement).
- To delete your account, email us. Payment records are kept as long as tax law requires.

Details: [Privacy Policy](#) and [Your data, in plain English](#).

15. Payments

Pro is paid in Stripe's embedded checkout. Card details go from your browser to Stripe and never reach our servers: we don't store, process or transmit card numbers. We keep Stripe's customer and subscription identifiers, status and period, and each payment's amount, currency, date and reference. Stripe's notifications are accepted only with a valid signature (checked in constant time, within 5 minutes), applied once, and the subscription is then read back from Stripe rather than trusted from the message. A payment is linked to

an account only through the account ID we put in the checkout ourselves, never through an email address.

16. Logging, monitoring and incident response

- **Security log:** sign-ins and failures, resets, two-step verification, PCs added and removed, team and settings changes, management actions and every relay session, with IP addresses where relevant (never for sessions), for 90 days. Pro accounts can download theirs.
- **Never logged:** passwords or derived keys, authenticator secrets or codes, note text, script text or output, clipboard contents, or anything seen or typed in a session.
- **At the PC:** notices and a local activity log give the person there their own record. **Modified apps** report themselves at sign-in.
- **Service health:** request counts, status codes and response times, sign-in failures and session counts, on our own metrics server.

Incident response: if a data breach is likely to cause serious harm, we'll tell those affected and the relevant authorities, as the law requires. Security questions: support@remoteaccessapp.com.

17. Reporting a vulnerability

Found a security problem? Email support@remoteaccessapp.com with "Security" in the subject, describing what you found, how to reproduce it and its impact. We'll confirm we received it, keep you updated, and tell you when it's fixed. Please:

- test only with your own account and PCs, and never access, change or keep other people's data;
- not degrade the service, for example with load tests or high-volume scanning;
- give us reasonable time to fix the problem before making it public.

We don't run a paid bug bounty. Our contact is also at [/.well-known/security.txt](https://well-known/security.txt).

18. Compliance status

RemoteAccessApp is made by Rust Games Australia Pty Ltd, an Australian company. Our [Privacy Policy](#) covers rights under the Australian Privacy Act and laws such as the GDPR, and our [Terms of Service](#) are governed by Australian law, including the Australian Consumer Law. We don't hold any third-party security certification or attestation yet, such as SOC 2 or ISO/IEC 27001; this paper describes our own design and testing. For a security questionnaire, email support@remoteaccessapp.com.

19. Known limitations and roadmap

- **Browser sessions aren't end to end:** the website server and its CDN are in the path.
- **The relay and account server are trusted:** a compromised relay could intercept a session someone at the PC then allows; a compromised account server could issue tickets and tasks, still subject to each PC's consent rules.
- **Code signing is in progress;** updates are verified with our own signature meanwhile.
- **Local-network links carry no tickets,** so team rights and plan limits apply only over the relay; the Console listens on a local-network port for them.
- **Encryption at rest uses a key on the same server;** backups are encrypted only once the owner's off-server key is installed; email and IP addresses are stored as they are.
- **No hard per-account lockout** (on purpose, so nobody can lock an owner out): wrong passwords pause the guessing network, plus two-step verification or email codes.
- **Public pages share the app's web address:** the chat and ad scripts there, sign-in and sign-up included, could act for a signed-in visitor if their vendor were compromised.
- **Powerful features:** the file browser reaches what the Windows user can, and the PC's log masks secrets in command lines only on a best-effort basis.
- **Older apps** (before 0.11.2) still send the password at sign-in.
- **One production server;** no self-service account deletion yet; no third-party certification yet.

Next: Authenticode signing, plan enforcement on local-network links, and requiring ticket-aware Agents once the fleet has updated. We'll update this paper as each lands.

20. About this document

Version 1.1, October 2, 2026. It describes RemoteAccessApp as it works today, and the copy at remoteaccessapp.com/security/whitepaper is always current. A shorter overview: [Security and privacy](#). Questions: support@remoteaccessapp.com.