BROADBAND TESTING

Atsign Report

Atsign AI Architect and the Atsign Platform examined

A Broadband-Testing Report

First published April 2026 (V1.0)

Published by Broadband-Testing

E-mail : info@broadband-testing.co.uk
Internet: HTTP://www.broadband-testing.co.uk

TABLE OF CONTENTS

BROADBAND-TESTING

Broadband-Testing is an independent testing operation, based in Europe. Broadband-Testing interacts directly with the vendor, media, analyst, consultancy and investment communities equally and is therefore in a unique space within IT.

Testing covers all aspects of a product/service from business rationalisation in the first instance to every element – from speed of deployment and ease of use/management, through to performance and accuracy.

Testing itself takes many forms, from providing due diligence for potential investors through to public domain test reports.

Broadband-Testing is completely vendor neutral and independent. If a product does what it says on the tin, then we say so. If it doesn't, we don't tell the world it does what it cannot do… The testing is wholly complementary to analyst-related reports; think of it as analysts getting their hands dirty by actually testing what's on the latest hype curve to make sure it delivers on its claims and potential.

Broadband-Testing operates an **Approvals** scheme which prioritises products to be short-listed for purchase by end-users, based on their successful approval, and thereby short-cutting the evaluation process.

Output from the testing, including detailed research reports, articles and white papers on the latest IT technologies, are made available free of charge on our web site at HTTP://www.broadband-testing.co.uk

EXECUTIVE SUMMARY

- Market research suggests that there were around 90,000 AI platforms in existence at the beginning of 2026 with, globally, around 35% of people now using AI tools daily and 90% in the technology world. That adds up to a massive number of AI apps (generative and agentic) being deployed and used daily, but how many are actually secure upon release – or indeed, ever?

- As data breach figures rise daily, the injection of non-secured AI apps into the public Internet, as well as private networks, is creating an ever-growing security problem, especially so when they are used in supply chain environments. For CISOs this is an authentic nightmare scenario. DevOps wants to create and release AI apps without underlying security protection; businesses need to compete at the highest level and that means embracing and using AI.

- The thought process on cyber security is still based around "reducing the attack surface" – a surface that AI apps are only increasing the size of by the second. Instead of this "band-aid" approach to security, an infinitely better alternative is to think of starting with "a zero-attack surface", only providing access to resource as and when required and, ideally, with a fully secure methodology behind it.

- With its Atsign Platform approach to ensuring every AI app is fully secure before release, regardless of the LLM (Large Language Model)/AI tool being used to create it, Atsign is fully embracing this "zero attack surface" concept with a genuinely unique approach and platform.

- In tandem with the Atsign AI Architect development tool and interface, it means that applications can be rapidly developed, without any initial coding whatsoever and then secured via the Atsign Platform before release, often within hours of first being conceived. This is revolutionary in that it makes AI development both ultra efficient and ultra secure.

- We looked at a sample application from a customer of Atsign's, KRYZ which, when initially developed manually took several weeks to build. It was then recreated from scratch in just one afternoon and – courtesy of the Atsign Platform - was able to be released immediately as a fully secure app. Moreover, since the Atsign Platform is LLM/AI tool agnostic, the developer was able to almost instantly recreate that app using a different LLM (first with Claude, then with Gemini – for the mobile front-end) by simply feeding the blueprint created with Atsign AI Architect into the second LLM. The same applies for supporting multiple endpoint device interfaces – for example, macOS and Android.

- For businesses looking to maximise their "ideation to adoption" process with AI, what Atsign is offering, in terms of zero attack surface and policy control, is an absolute game changer. For CISOs, it means their AI hell is at and end; just use the Atsign Platform and every app will be secure before release. Using the combination of the Atsign Platform with Atsign AI Architect means that the entire dev process is hugely streamlined and optimised – and secure. The holy trinity? You'd better believe it!

MAKING AI SECURE

The IT world gets many things right, but it also gets a lot of things very wrong.

Top of that "very wrong" list right now would have to be security. Since the mid-80s, companies have been applying a bolt-on approach to security, with the result that most network deployments have more holes than a Swiss cheese. In very few scenarios did security form the foundations of those deployments; instead it has been built as almost a secondary infrastructure, on top, inside, outside and generally "all over" that underlying data and application transport mechanism, often laughingly known as "infrastructure". This isn't being mean – it is exactly the reason why breaches are occurring daily and increasing by the day.

And now we have the veritable onslaught of AI applications being injected into those already fragile infrastructures. It is reasonable to say that most DevOps folks aren't exactly security-forward in their thinking. Nah – security, that's the boring bit. Someone else can add that. Worse still, many of the AI applications are designed to be part of a supply chain kind of infrastructure; one that, in theory, optimises delivery by removing the "slow human" element but, if unsecured, creates two-way targets for cyber criminals. Target the weak link and they can then go up and down that chain, probably using AI agents to form the basis of their attack team. There's an irony…

Unsurprisingly, therefore, the world of IT social media is crammed full of scare stories concerning the cyber security risk that AI applications are injecting into the network and how to best "reduce that attack surface". Even the AI 'fathers' such as Anthropic are now accepting that there is a major security risk and attempting to be pro-active about it. In this case, enter Claude Code Security, currently available in preview form and designed to scan codebases for security vulnerabilities and suggest targeted software patches for human review. It sounds remarkably like a basic detection scanner for AI code, which is hardly proactive and is still using the "reduce the attack surface" mentality as its provenance.

So, how about – instead - thinking in terms of "it's not a case of reducing your attack surface, but about starting off with absolutely no surface whatsoever to attack in the first place". In other words, start with a zero-attack surface and only open up, as and when those resources are required? Better still, how about creating a secure transaction technology that means nothing has to be open? If it doesn't appear to exist – let's say a TCP port – then how do you attack it? Spoiler alert – you can't. Back to the AI world; what if this concept could be applied natively to every AI application before it is released into the world of bad actors, AI double agents and fake credentials? Such is the thinking being Atsign, the focus of this report, along with the ability to optimise time to delivery of those AI applications in the first place. A potential double whammy for the AI cybercriminal community? You bet.

Let us then, firstly describe what Atsign has built and then see it in action.

ATSIGN OVERVIEW

Atsign has created two fundamental elements forming a complete AI development and deployment solution; first, with Atsign AI Architect you have a developer tool that provides the necessary blueprint and instructions for designing AI-coded apps.

Underpinning this is the Atsign Platform - an open-source, full-stack platform for developing applications and services with integrated security, privacy, and data control. Put simply, it enables end-to-end, encrypted communication between any two parties - an Internet-connected person, entity, or thing – without the need for any additional network setup. All the security elements such as encryption, authentication, and certificate management are automatically managed by Atsign Platform.

Combine the two and you have a complete architecture for creating and delivering fundamentally secure AI apps in a fraction of the time it would take to create and secure these manually. And there is no lock-in to a specific LLM/development tool – it is completely agnostic in this sense.

Atsign AI Architect

Put simply, as the company itself says, Atsign Ai Architect provides the ability to generate accurate, integrated, and secure code for any application or service.

It is constructed around a three-stage process: create the app blueprint through a drag and drop interface, specify the instruction set by compiling the design into a rigid JSON specification to create the LLM prompt and finally orchestrate that production build by feeding that specification into whichever LLM/AI tool you wish to use, whether that's Claude, ChatGPT. Gemini or whatever your preferred model is.

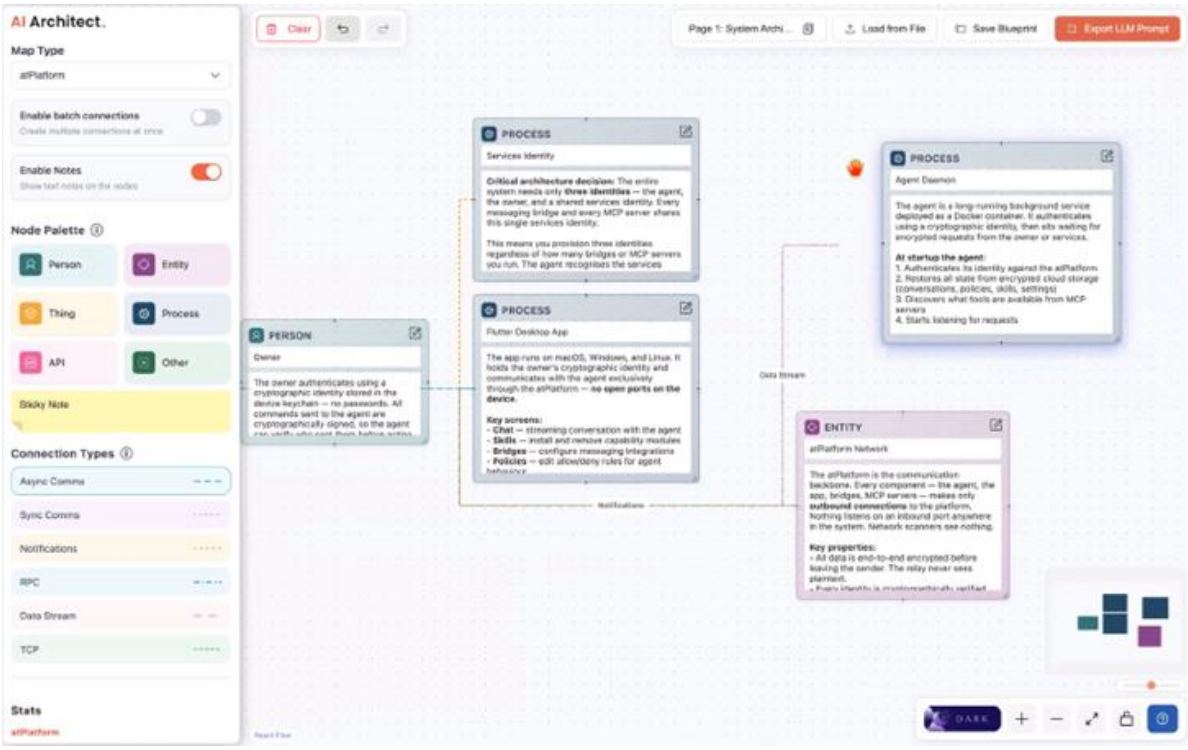

Figure 1 – The Atsign AI Architect User Interface

Creating the blueprint means visually defining your system's topology and data flows: actors, data flows, and permissions – the app as a model, in other words, with absolutely zero coding involved. Atsign AI Architect then compiles that blueprint design into exportable JSON format, thereby creating your LLM prompt. Key here is in creating a very tight, deterministic instruction set that leaves zero room for an LLM to add its own interpretations – and thereby getting it wrong!

Effectively, what it creates is a "digital contract" in terms of defining the deterministic instruction set for an engineering team or LLM: fully defined and architecturally rigorous. It is impossible to overstate the importance of not providing any means for an LLM to misinterpret – or override – that code, if it is to be production ready, every time, rather than spending hours and days back-tracking through and editing the code in order for it to be usable in the real-world.

At this point, that specification is now ready to be fed to an LLM; the AI generates production-ready code that is secure by design from line one, courtesy of Atsign Platform's secure networking infrastructure, all but invisible though it is in operation. Moreover, regardless of the nature of the app or service, the A-Z procedure is exactly the same every time; from a user perspective, only the model content differs.

Atsign Platform

The Atsign Platform is built around four primary elements:

Atsign Platform Protocol: this is a secure, application-layer protocol that sits on top of TCP/IP, enabling peer-to-peer data exchange between any two endpoints, ensuring end-to-end encryption and data integrity.

Atsign SDK: the SDK provides Atsign Platform-specific building tools, allowing developers to optimise application builds and/or embed the Atsign Platform protocol into existing software or device firmware.

Atsign: This is a cryptographically verifiable, unique identifier and address that can be owned by individuals and entities and assigned to things and AI agents. For example: @bob, @alice, @camera219.

Atsign Server: this is a personal data service for storing and exchanging encrypted data owned by an Atsign, as well as a place for information exchange. So, each Atsign has a corresponding Atsign Server in the form of a one-to-one relationship.

One standout – and very important – difference between a classic client-server based encrypted architecture and the Atsign Platform is that, with the former, the server still receives all the data as it passes from end-to-end, thereby creating a very obvious attack surface, in addition to the use of firewalls and other devices that may be misconfigured and reveal vulnerabilities from open ports. But in the case of the Atsign Platform, the servers never see the data and have no encryption keys to access it. This is the benefit of the 1:1 design of Atsign – only the two private parties are aware of each other.

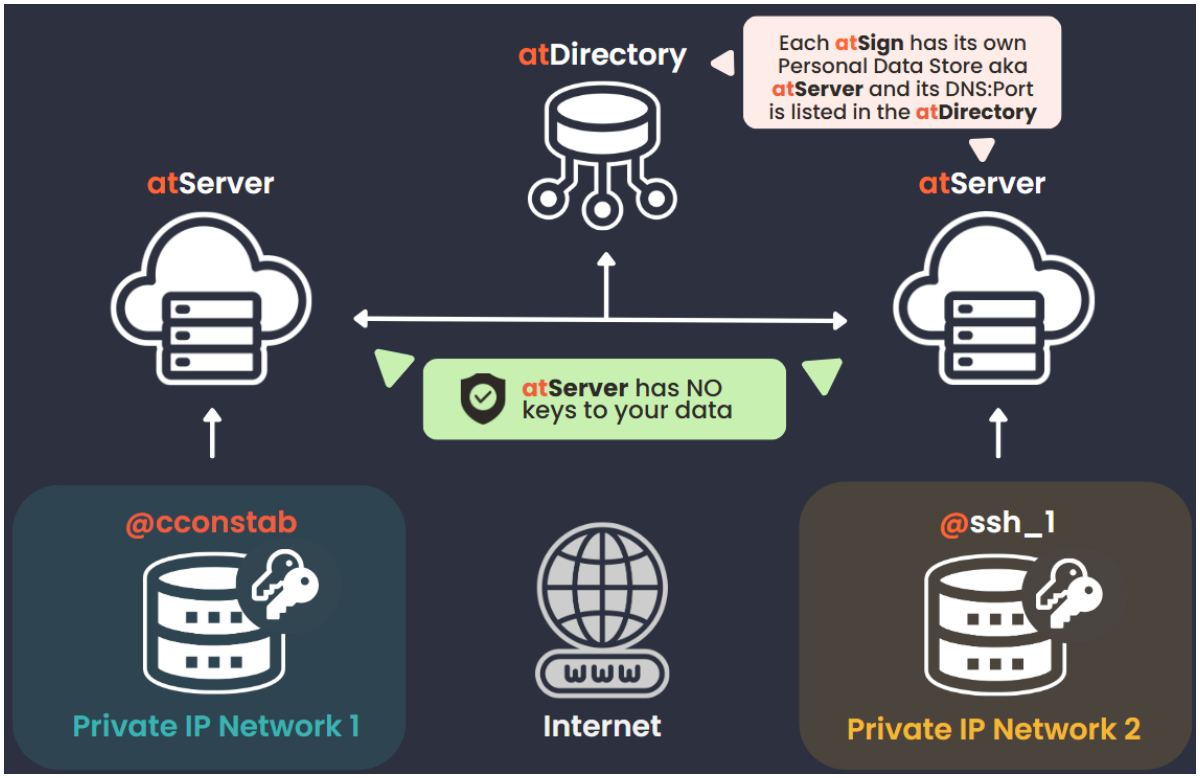

Figure 2 – Atsign Servers Cannot See The Data

With Atsign Platform, devices don't require open ports for communication, meaning common attack vectors such as port scanning are totally ineffective. Only authorised parties can find and connect to devices.

Atsign likes to use a "post office box" analogy in order to explain how the Atsign Platform works, as follows:

- Give a package a unique address (Atsign). This is its destination.
- The Atsign SDK automatically encrypts the package contents and stores it in the sender's postal box (Atsign Server)
- The Atsign Platform protocol provides the delivery rules to notify the receiver's post office box.
- The receiver's postal box picks up a clone of the package.
- The recipient uses their Atsign to retrieve it from their postal box and the Atsign SDK automatically decrypts their package contents.

And, as the company themselves would add, being a digital service, unlike any standard post office around the world, in this case the delivery is always made. Instantly and securely.

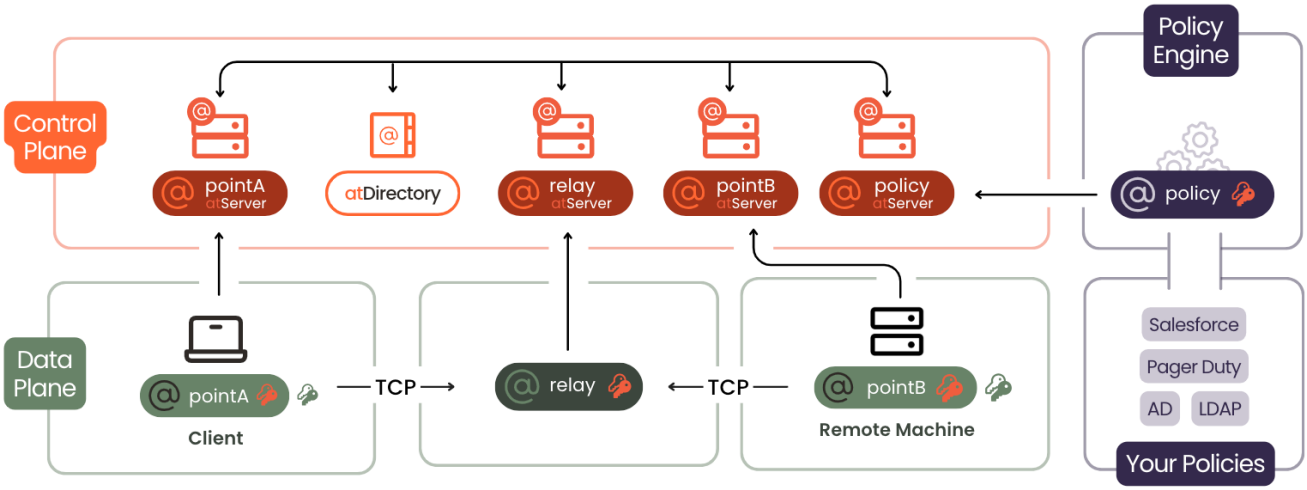

Figure 3 – The Atsign Platform

NoPorts

In addition to the actual Atsign Platform, Atsign also offers the NoPorts solution for remote access, using the same "closed-by-default" ports concept – hence 'NoPorts'.

It provides encrypted, direct communications with no centralised or reverse proxies or overlay network, no vendor-controlled keys or certificates, no VPNs and no port management or static IPs. Basically NoPorts uses the Atsign Platform and Atsign to create an encrypted connection between devices over TCP/IP. However, TCP ports are not open on the endpoints, so the connection is completely invisible to potential attackers as there is no entry-point.

Any device or application sitting behind NoPorts has – literally - no open ports, no static IP address requirement and cannot therefore be digitally hacked, thereby able to securely exchange information without any possibility of surveillance, impersonation, or theft. Any TCP application can be set up to utilise a NoPorts connection and the encrypted tunnel is set up with outbound requests only which are sent to an Atsign Server, responsible for managing identity and maintaining the key-value data store for each Atsign.

A request to the NoPorts relay service results in it allocating a pair of ports on the host it is running on and an IP address, which the client receives. The client then sends a request to the remote endpoints to connect. The request includes session information, the client's intent (such as destination TCP port 3389 on localhost), and the generated public key from an ephemeral asymmetric key-pair. If granted via the NoPorts policy service, a connection is made, otherwise it is refused. Simple as…

ATSIGN IN PRACTICE – A REAL-WORLD USE CASE

To see Atsign's architect and platform technology in action, we looked at a real-world application which took a fundamentally non-secure data stream (SNMP – Simple Network Management Protocol) and fully secured it while being able to access statistics from a remote radio station.

Given the ongoing usage of SNMP globally in its various release formats, this is a great example of, not only demonstrating the creation and deployment of a secure AI app, but also how fundamentally non-secure – but extremely useful - data can still be used in 2026 and beyond.

Use Case Overview

KRYZ-LPFM is a low-power FM radio broadcast station whose Nuatel VS series transmitter exposes its operational telemetry — modulation, SWR, forward and reflected power, heat-sink temperature, and fan speed — through SNMP, a 1980s-era protocol that runs on UDP and was never designed to leave a trusted network.

The question was straightforward: how does the station owner check on the transmitter from anywhere in the world, on a phone, without turning the station into an attackable asset on the public Internet?

First, let us clarify just how non-secure the SNMP data is.

- It runs on UDP, typically port 161. UDP is connectionless; packets can be spoofed, replayed, or forged with no handshake.
- SNMPv1 and v2c authenticate with a "community string" sent in the clear – totally unencrypted. So anyone with a basic packet capture app on the path effectively 'owns' the device within seconds. Note: with v3 of SNMP, encryption was added but many deployments – such as the transmitter here – predate it, or were simply not able to support it in a manageable way, as it was both resource hungry and clunky at best.
- SNMP assumes a trusted network, not least because the original version (1988) predates many, now common, network security technologies.

So, without creating a secure, AI-based tool to remotely monitor the transmitter securely, what were the historical options available?

Expose SNMP directly: port-forward UDP/161 to the transmitter. This publishes the device, its community string, and its telemetry to the entire Internet. Any shodan.io scan will find it within hours at most.

Run a VPN: this means deploying a VPN concentrator at the station, managing credentials, certificates and client configuration for every user. In so doing, it effectively replaces one attack surface with another - the VPN endpoint itself, as well as adding significant ongoing operational costs and potential management nightmares.

Poll from a cloud monitoring service: in this scenario the service still needs a path into the station, so the firewall rule or VPN requirement do not go away; you're simply *moving* the problem, not *removing* it.

Please note: All three options share one feature - the station has to have a publicly-reachable listening port somewhere. That port is, in itself, an attack surface, so this had to be eliminated from the equation.

The Creation Process

First came the monitoring software design, courtesy of Atsign AI Architect, ultimately feeding into Claude as our chosen LLM.

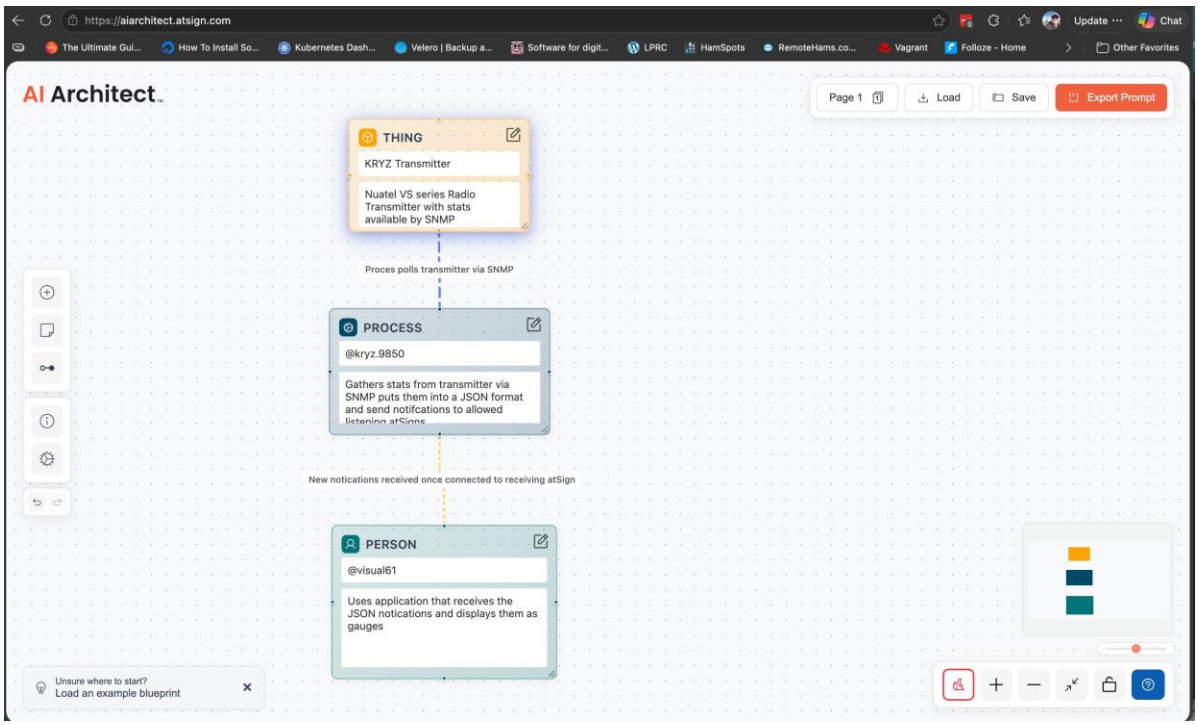

Figure 4 –The KRYZ Blueprint.

As described earlier, Atsign AI Architect allows you to simply create, drag and drop style, a flow combining three "elements: 'Thing', 'Process' and 'Person'. In the case of KRYZ, this was the basis of the blueprint – containing three nodes and two edges - created:

KRYZ Transmitter (Thing) — the Nuatel VS series transmitter with its SNMP agent.

`@kryz.9850` (Process) — the Dart daemon that polls the transmitter via SNMP, assembles JSON, and sends notifications to permitted listening atSigns.

`@visual61` (Person) — the operator, running an app that receives the JSON notifications and renders them as gauges.

Edge 1 — a **stream** from the transmitter to the collector process, representing periodic SNMP polling.

Edge 2 — a **notification** from the collector to the app, representing encrypted push of new readings.

Blueprint Before Coding: Observations From Atsign CTO Colin Constable

There is a reason the blueprint is drawn before a single line of code is generated and it is the reason this entire approach works. LLMs generate plausible code from plausible prompts. Given a vague request — "build me a radio monitoring app" — they will invent an architecture, usually one that looks like every other web app in their training data: a REST API, a cloud database, an authentication token and a dashboard. That architecture, by definition, has a publicly-reachable server. It has an attack surface.

The blueprint forecloses that option. It captures three things that the LLM would otherwise have to guess: who the actors are, how they communicate, and what identities they use. Those decisions are made once, visibly, by a person, and then handed on to the next stage of the pipeline as non-negotiable constraints.

A Note on "vibe coding": Atsign uses the term *vibe coding* for the practice of prompting an LLM to build systems from loose natural-language requests. Atsign AI Architect is their answer to it. The blueprint is the contract between the designer's intent and the generated code — it turns a conversation into a specification. The CISO gets something to review *before* the build starts, not after.

The basics of the creation methodology here were simple: create the idea and turn it into the blueprint, which then partially creates the LLM prompt, which is then turned into a secure app (or it could be an agent).

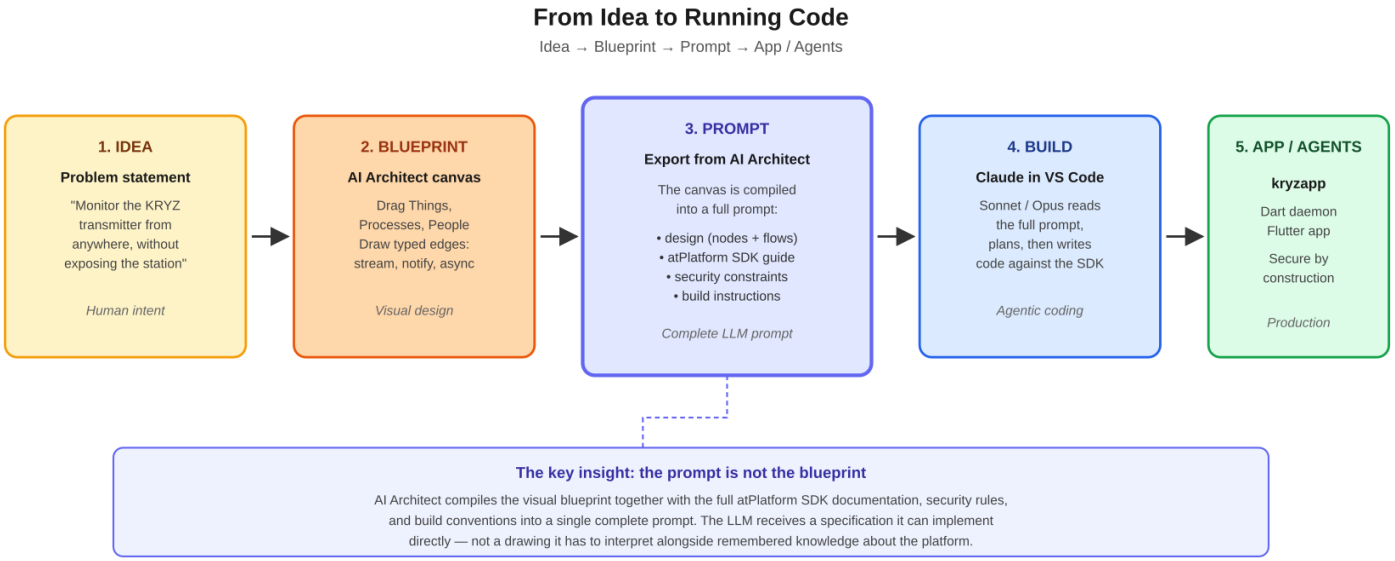

Figure 5 –The KRYZ Blueprint As Drawn On The Atsign AI Architect

1 — The aim: To monitor the transmitter from anywhere, without exposing the station. At this stage there is no diagram and no code — only a requirement and a set of constraints that any solution has to satisfy.

2 — The blueprint: created in Atsign AI Architect by dropping nodes onto the canvas: one 'Thing' for the transmitter, one 'Process' for the collector and one 'Person' for the operator. Typed edges connect them. The canvas handles the bookkeeping — each node gets a unique ID, each edge records its source, target, and connection type and the whole graph is validated for consistency. Annotations attached to nodes and edges capture the designer's intent.

Looking at the actual saved blueprint file, each node entry records its type, label, notes and position. The relevant fragment for the collector process is:


```
{
  "id": "node_1764827709043_sxcjo2pbh",
  "type": "custom",
  "data": {
    "label": "@kryz.9850",
    "type": "process",
    "notes": "Gathers stats from transmitter via SNMP, puts them
              into JSON format and sends notifications to
              allowed listening atSigns"
  }
}
```


The blueprint names each participant with its atSign — @kryz.9850 for the daemon, @visual61 for the user. Those are not placeholders; they are the cryptographic identities that the generated code will use at runtime. The design and the identity scheme are the same artefact.

3 — Prompt (the hinge): easy to miss if you have only seen the canvas. Note: the blueprint is not what Claude receives. What Claude receives is a full prompt that Atsign AI Architect produces by combining the blueprint with everything else an LLM needs in order to generate working Atsign Platform code.

That prompt contains:

- **The design itself** — nodes, edges, identities, and connection types, drawn from the saved blueprint.

- **Atsign Platform SDK documentation** — concrete instructions for using the `Atclient` SDK to authenticate, send notifications, subscribe to updates, and manage encryption keys.

- **Security conventions** — rules like "initiate all connections outbound," "authenticate cryptographically before exchanging any data" - "use Atsigns as the only identities" and "do not expose listening ports on any component."

- **Build instructions** — preferred language choices (Dart for daemons, Flutter for the app UI), project structure and the expected separation of concerns between daemon, app, and shared model code.

Note: the LLM is not being asked to remember how the Atsign Platform works, or to guess at the right SDK calls, or to invent a plausible architecture. It is given a *precise* specification plus the *exact* reference material needed to implement it. Hallucination risk drops sharply, because the most hallucination-prone parts of the task — API shapes, security patterns, conventions — are supplied in the prompt rather than recalled from training.

Why this matters

A blueprint alone, handed to an LLM, would still leave a lot of room for the model to improvise: for example, *how* is a "notification" actually implemented and which SDK methods are called, or what does authentication look like? Atsign AI Architect answers those questions **inside the prompt**, so the model does not have to. The result is code that compiles against the real SDK and runs on the real platform, not a plausible approximation of either, which potentially fails to deliver the intended results.

4 — Build: the compiled prompt is pasted into a Claude Sonnet or Opus session inside Visual Studio Code. From there, a standard agentic workflow takes over: Claude reads the prompt, produces an implementation plan, asks clarifying questions where the blueprint is silent (for example, the SNMP polling interval where five seconds was chosen in this case, or the specific OIDs for the Nuatel VS's telemetry), and then writes the code directly into the project tree.

Since the prompt already names the Atsigns, the SDK, and the connection types, there is very little room for Claude to improvise. It does not invent a REST API, because the prompt does not describe one. It does not reach for a SQL database, because no node is marked as a datastore. It builds *exactly and only* what the prompt describes, using the Atclient SDK as the transport because that is what the connection types compile to in this ecosystem.

5 — App/Agents: the output is two Dart/Flutter projects sitting in a shared repository, each with its own Atsign:

`snmp_collector/` — a headless Dart daemon that runs on any machine with network access to the transmitter (typically a small box inside the station LAN, like a Raspberry

Pi). It polls via SNMP, encodes readings as JSON, and calls `atClient.notificationService.notify()` for each update.

`mobile_app/` — a Flutter application (here shown running on macOS, but portable to iOS and Android) that onboards with the operator's Atsign, subscribes to notifications matching a regex, and renders readings as Syncfusion gauges with configurable alert thresholds.

`shared/` — a small shared Dart package that defines the `TransmitterStats` model and the Atsign configuration, so both the daemon and the app agree on the JSON schema.

The Finished App: The Working Runtime Architecture

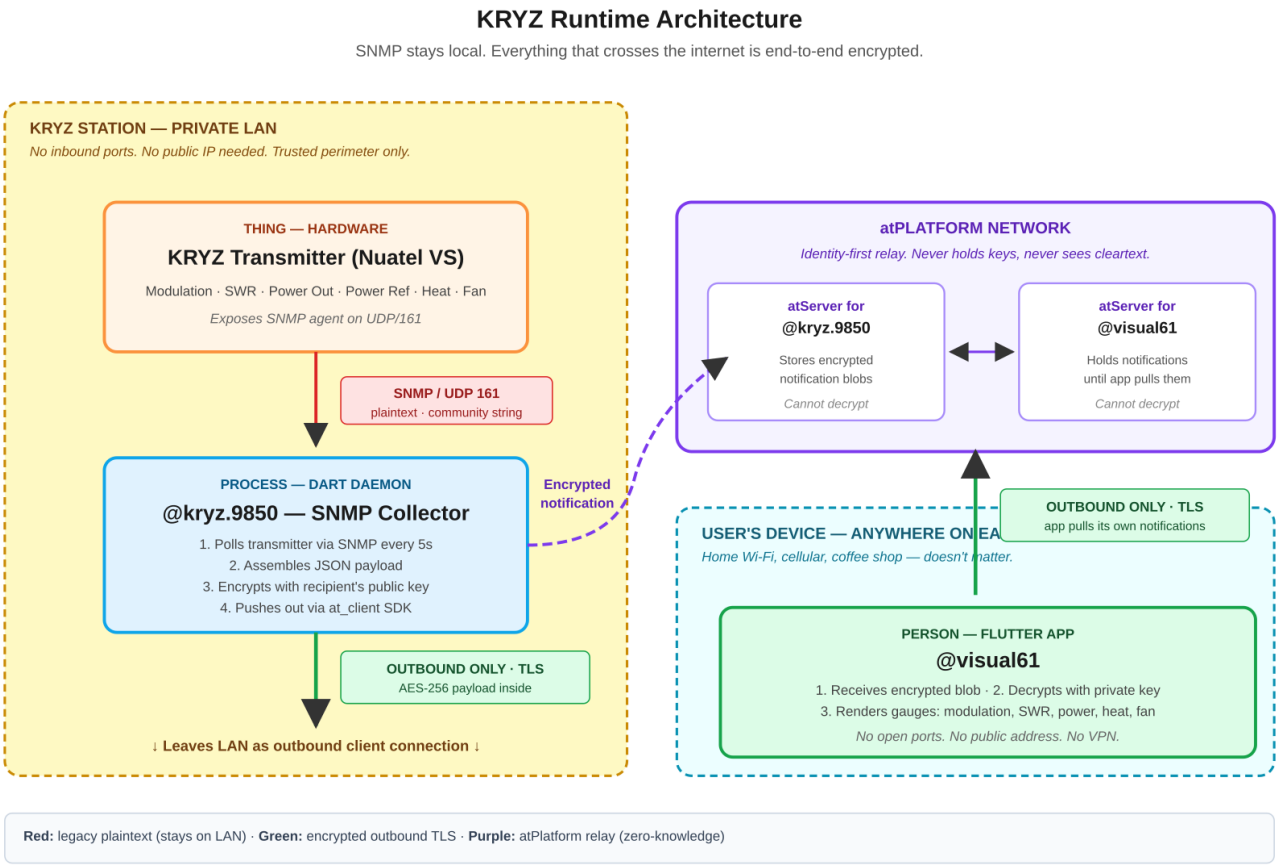

Figure 6 – The Runtime Architecture

At runtime, KRYZ has three tiers separated by clear trust boundaries. The SNMP side of the system never leaves the station's local network. Everything that crosses the Internet is end-to-end encrypted and carried over connections that the station and the user's device both initiate outbound.

Tier 1 — the station LAN

Inside the station, nothing has changed. The Nuatel VS transmitter continues to speak SNMP on UDP/161 exactly as it always has. The SNMP collector daemon sits on the same LAN as the transmitter, polls it every five seconds, and does its work entirely within the trusted perimeter. The transmitter has no idea the Atsign Platform exists, and it does not

need to. Note: all this matters, as it means the integration works with any SNMP-capable device and no firmware changes are required. In other words, legacy equipment that will never see another security patch gets modern security by virtue of never being exposed in the first place.

Tier 2 — the Atsign Platform relay

The collector daemon initiates an outbound TLS connection to its own Atsign Server — the tiny cloud-hosted storage endpoint associated with the @kryz.9850 atSign. The daemon authenticates cryptographically using its private key (which lives only on the collector machine) and pushes encrypted notifications. The Atsign Server stores these notifications and, when the recipients' Atsign Server asks for them, delivers them. The Atsign Server pair is the only component of the system that is Internet-reachable, and the data they hold is cryptographically opaque. The relay operates on a zero-knowledge basis: the Atsign Platform's operators cannot read the payload, cannot derive the keys, and cannot impersonate either Atsign. The network is a carrier, not a trustee.

Tier 3 — the user's device

The Flutter application on the operator's device initiates its own outbound TLS connection to the @visual61 Atsign Server and pulls pending notifications. It decrypts them locally, parses the JSON, and updates the UI that was shown in Figures 5 and 6. The device never listens for inbound connections; it never has a public address; it works identically on home Wi-Fi, LTE, a hotel network, or a coffee-shop hotspot, for example. The location and network posture of the device are invisible to the station.

The Security: End-to-End Encryption. Who Can Read What

The Atsign Platform uses standard, well-understood primitives — AES-256 for payload encryption and RSA for key wrapping — arranged so that the relay servers cannot decrypt what they carry.

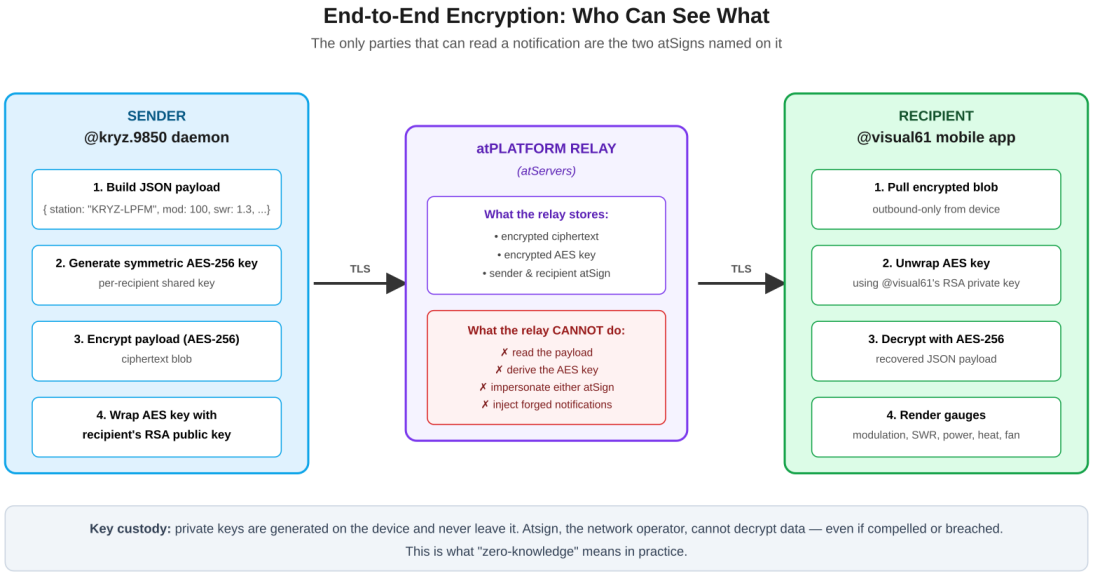

Figure 7 – Single Notification Travels From Collector To App

Step by step Description

- **The sender assembles the payload.** The collector daemon builds a JSON object — modulation, SWR, forward and reflected power, heat, fan speed, timestamp — that describes the current state of the transmitter.
- **A fresh AES-256 key is generated for the sender-recipient pair**. In the Atsign Platform model, each pair of communicating Atsigns has its own symmetric key, established the first time they exchange data.
- **The payload is encrypted with AES-256 using that shared symmetric key**. The result is a ciphertext blob that is indistinguishable from random bytes without the key.
- **The symmetric key itself is wrapped with the recipient's RSA public key.** The wrapped key is sent alongside the ciphertext, so the recipient — and only the recipient — can unwrap it.
- **The encrypted package transits the Atsign Platform.** The @kryz.9850 Atsign Server stores it, the @visual61 Atsign Server receives it, and neither can do anything useful with it.
- **The app pulls the package over outbound TLS.** It unwraps the AES key with @visual61's RSA private key (which has never left the device), decrypts the ciphertext, and parses the JSON.

What this rules out...

This design places concrete, testable limits on what any attacker — including an insider at Atsign — can accomplish:

- **The platform operator cannot read the data.** They hold no keys capable of decryption. Keys are generated on user devices and never transmitted.
- **The platform operator cannot impersonate either party.** Authentication is via cryptographic signature using the Atsign's private key, which only the legitimate owner holds.
- **The platform operator cannot inject forged notifications.** Each notification is signed; a forged one would fail signature verification on the recipient.
- **A compromised Atsign Server leaks only encrypted blobs.** Even a complete dump of the relay's storage yields ciphertext, not telemetry.
- **A compromised Atsign affects only that Atsign.** Each identity's blast radius is limited to the data that identity has access to. There is no central credential store to breach.

Attack Surface: What Can Actually Be Reached?

It is equally important to analyse what might still actually be attacked and potentially breached, so the analysis enumerates every network-reachable component of the running system and asks, for each, what a remote attacker with arbitrary resources and no prior access could do.

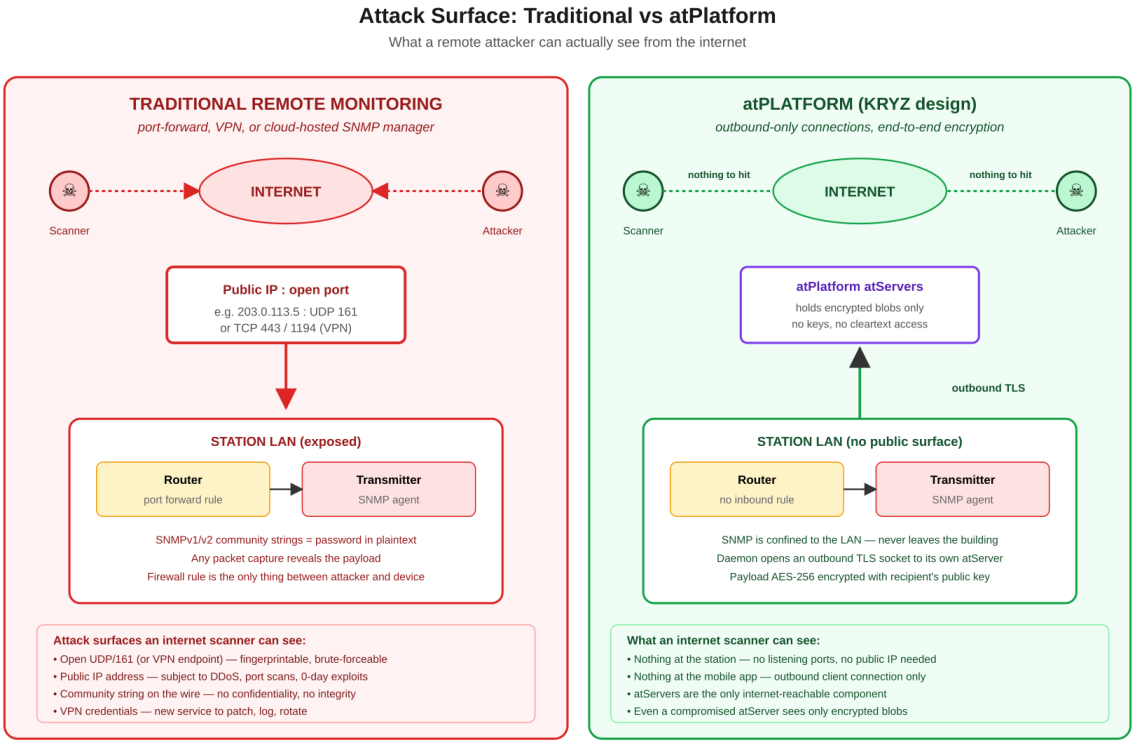

Figure 8 – Attack Surface Comparisons

The station-side attack surface

Note: traditional remote monitoring — port-forwarded SNMP or a VPN concentrator at the station — publishes at least one listening endpoint on the public Internet. A scanner (Shodan, Censys, or any consumer tool) will find it and fingerprint it. Once found, that endpoint faces the full catalogue of Internet risks: zero-day vulnerabilities, credential brute-forcing, denial-of-service, protocol confusion, and whatever else the next decade produces.

In the KRYZ deployment, the station has *no* public listening port. There is *no* port-forward rule on the router. There is *no* VPN service. The router's inbound firewall can, in principle, be set to deny everything. A scanner targeting the station's IP range sees closed ports on whatever the ISP has given the router — indistinguishable from any residential or small-business connection with nothing running behind it.

The collector daemon only makes outbound connections. From a network perspective, it behaves like a web browser: it opens a TLS socket to a known server (the @kryz.9850 Atsign Server), uses mutual cryptographic authentication, and pushes data. There is nothing for an attacker to send packets to.

The device-side attack surface

The operator's device is in exactly the same posture. The Flutter app makes outbound connections only. It does not run a background service that listens for inbound pushes (it does not need to — the Atsign Server holds notifications until the app asks for them). The device can move between networks, sit behind carrier-grade NAT, or run on airplane Wi-Fi, for example, and the security model is unchanged.

The relay-side attack surface

The Atsign Servers are the only Internet-facing components of the system. They are worth attacking — in the sense that they are reachable — but they are not worth much to an attacker, because:

- They hold only encrypted blobs.
- They authenticate every client connection by cryptographic challenge-response against the client's Atsign identity, before any data is exchanged.
- They do not store long-term credentials for clients — authentication is per-connection, proved by control of a private key the server does not hold.
- A successful compromise of an Atsign Server yields ciphertext, not cleartext; the attacker still needs the recipient's private key (which is on the recipient's device) in order to make sense of anything.

Comparison table

The concrete differences between a traditional deployment and KRYZ are summarised below.

Property	Traditional remote SNMP	KRYZ on Atsign Platform
Public listening port at station	Yes (UDP/161 or VPN endpoint)	None
Public IP/static DNS required	Yes	No
Inbound firewall rules	Required, must be maintained	None — default deny works

Property	Traditional remote SNMP	KRYZ on Atsign Platform
Credentials on the wire	Community string (cleartext in v1/v2c)	None — cryptographic auth only
Payload confidentiality	None (v1/v2c) or optional (v3)	AES-256 end-to-end, always
Credential store to breach	Yes (VPN/SNMP users DB)	None — no central credentials
Attacker scanning public IP	Finds endpoint, fingerprints it	Finds nothing at all
Revoking a user	Rotate credentials, reconfigure	Remove Atsign from allow-list
Relay operator can read data	Not applicable (no relay)	No — zero-knowledge

So, can the data actually be breached?

Taking 'breach' to mean "an unauthorised party obtains the transmitter's telemetry in a form they can read," the answer is: only through compromise of an endpoint that legitimately holds the data, specifically:

Compromise the collector daemon. An attacker who gains code execution on the collector machine can read the SNMP data directly from the transmitter. This requires first breaching the station LAN — the same as any on-premise attack.

Compromise the operator's device. An attacker who owns the device can read decrypted notifications because the device holds the private key. Again, this is a local-device risk that exists for any app holding data.

Steal a private key. If an attacker can exfiltrate the private key from either the collector or the device, they can impersonate that Atsign. Key material is stored in OS-level secure storage (Keychain on macOS/iOS, Keystore on Android, file-system protection on servers); extracting it requires at least root access on the target device.

Note: There is no remote attack against the Atsign Platform relay that yields cleartext data, because the relay has no cleartext data. There is no remote attack against the station, because the station has no remote-reachable service.

Is this "normal" security, or something new?

It is normal primitives — TLS, AES-256, RSA, atomic signed messages — composed in an unusual way. What is different is architectural. In conventional designs, security is a *property of the endpoints:* firewalls, WAFs, rate-limiters, and patch cycles are deployed around a publicly-reachable service to keep it safe. Security is a continuous effort against a surface that exists by default. In the Atsign Platform model, security is a *property of the topology:* there is no publicly-reachable service at the station or at the user, so the surface does not exist in the first place. The only Internet-reachable component (the Atsign Server) is cryptographically constrained to see only ciphertext. The attack surface isn't *minimised*; it is structurally absent.

Atsign calls this 'Zero Exposure'. That is a marketing term for a real engineering property: any system built on this topology inherits the property whether its designer was thinking about security or not. KRYZ is a small, concrete example of what that property looks like when applied to a problem — remote SNMP monitoring — that has historically been a case study in how to get network security wrong.

Using The Finished App

Here we are looking at the finished, live application from the KRYZ-LPFM deployment, starting with the blueprint and the app itself.

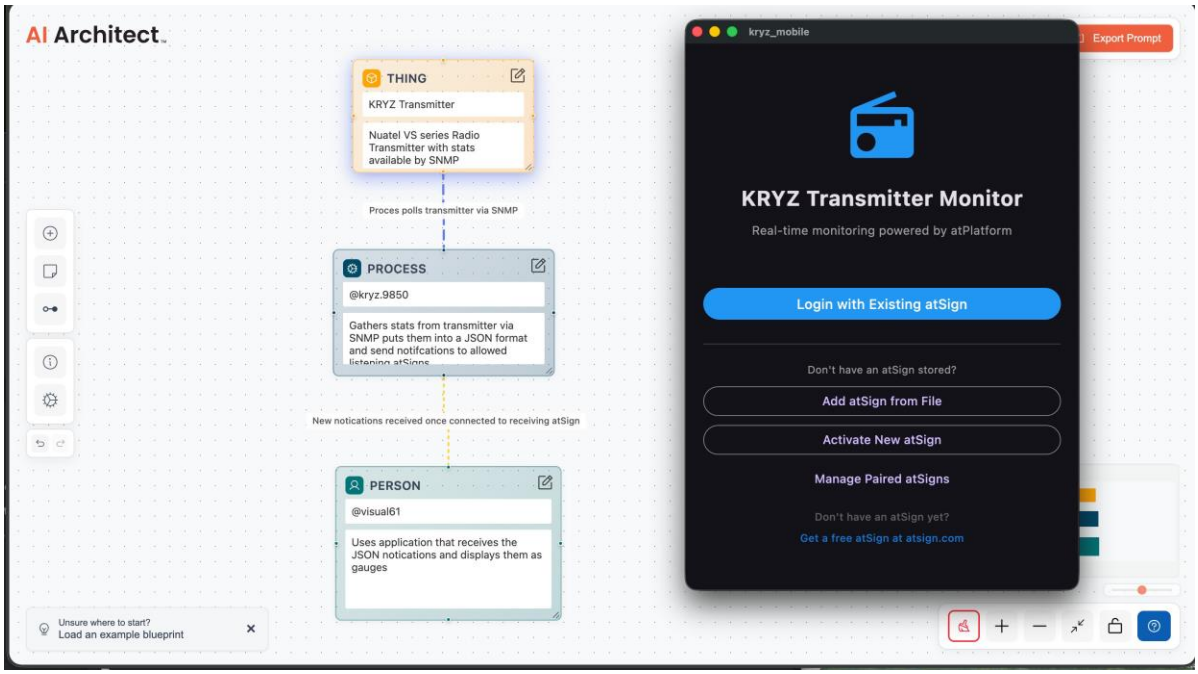

Figure 9 – Blueprint And App Side-By-Side

The KRYZ blueprint is visible on the left in the Atsign AI Architect tab and on the right is the compiled Flutter application connecting to Atsign Platform – from canvas, through export, to build session.

Onboarding: There are no usernames and passwords in this application. The operator picks an Atsign from the set of identities registered on the device, the app proves possession of that Atsign's private key to the Atsign Platform and that is the whole onboarding flow. This screenshot shows the selector with three Atsigns available on the machine — @cconstab, @colin, and @visual61. Choosing @visual61 loads the identity that was named in the blueprint and allowed to receive notifications from @kryz.9850. There is no "sign up", no email verification, no password reset path. The cryptographic identity *is* the account.

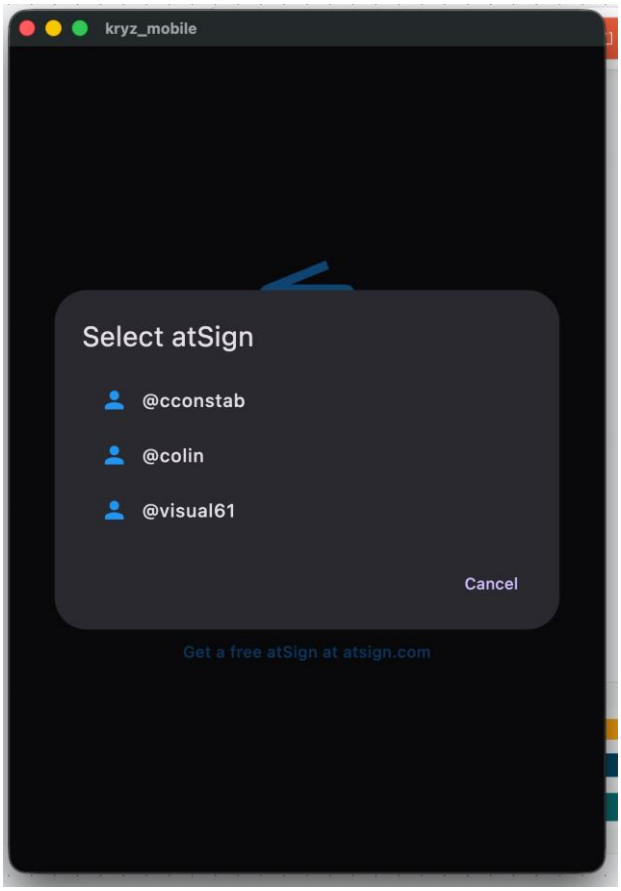

Figure 10 – App: Onboarding

The live dashboard: once onboarded, the app subscribes to notifications from @kryz.9850 and renders them. The screenshot shown next is a live dashboard at 17:48 on 20 April 2026, showing:

Station banner — "KRYZ-LPFM, ON_AIR," green. Status is lifted directly from the transmitter's SNMP MIB.

Modulation — 100.0%. The audio program is fully modulating the carrier.

SWR — 1.3:1. The standing wave ratio is healthy; any value climbing past ~2:1 would indicate an antenna or feedline problem.

Power Out/Power Ref — 11.0 W forward, 0.2 W reflected. This is an LPFM (low-power FM) station; the expected output is in the tens of watts, not kilowatts.

Heat Temp — 36.3 °C. The power amplifier's heat-sink temperature.

Fan Speed — 7321 RPM. The cooling fan is running, which lines up with the modest heat reading.

Note: the identity of the current user (@visual61) is visible top-right. Everything on this screen came through the Atsign Platform as an encrypted notification and was decrypted locally on the device.

Figure 11 – App: The Live Dashboard

Configurable thresholds: The gauge colours — green for normal, amber for warning, red for critical — are driven by per-gauge thresholds that the operator can tune. The Gauge Settings screen makes this explicit: each metric has a scale range and two thresholds (warning below critical). This is a purely local UI concern — it does not change anything on the transmitter, and it does not affect what the Atsign Platform carries.

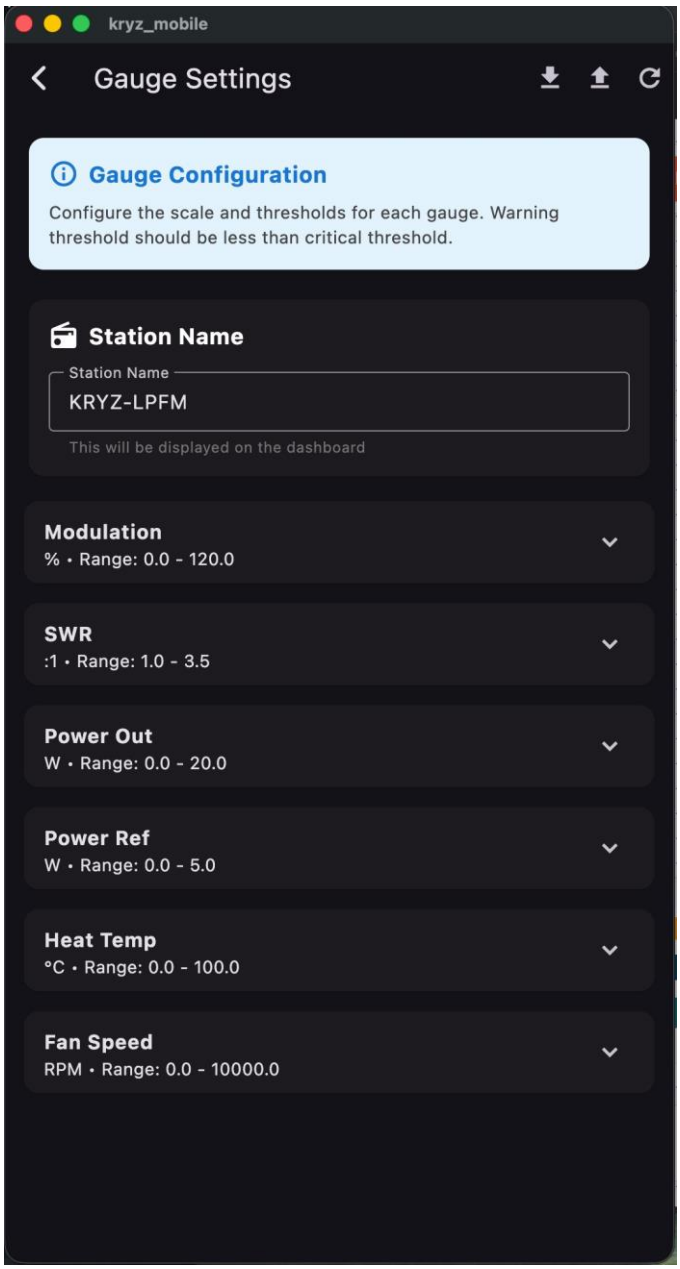

Figure 12 – App: Configurable Thresholds

IN CONCLUSION

Ai app development and adoption have gone beyond the typical analyst groups' hype curve peak and kept on rising!

Unfortunately – and terrifyingly so for CISOs – securing those generative and agentic apps has not exactly been top of the list of "to do" tasks before sending said apps out into the wild. This is sugar coated ether candy for the cyber attackers, especially when those apps are in supply chain environments. But businesses are under pressure to maximise the "AI moment" and gain that age-old competitive edge over their rivals, while DevOps teams simply want to pump out more and more AI apps.

Fortunately, Atsign has come up with a definitive – and game changing – solution to this crisis (yes, it's not a problem, it's a crisis alright) with the Atsign Platform to secure those apps – regardless of preferred LLM – and, with the made to match Atsign AI Architect, a simple and rapid way of creating and deploying those AI apps securely and consistently. The Atsign Platform approach can literally ensure that every AI app is fully secure before release, conveying the new stance on security: start and finish with a zero-attack surface.

We said "game changer" but, for businesses awaiting their impending "breach doom" as they try desperately to reduce their existing attack surface, it could just be a life saver.

And, if you are one of those companies longing to embrace the AI era but are – rightly – worried about the security implications of doing so, then what are you waiting for? The solution is right here, right now.

APPENDIX 1: ATSIGN PLATFORM AND MITRE ATT&CK

MITRE ATT&CK® is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations. The ATT&CK knowledge base is used as a foundation for the development of specific threat models and methodologies in the private sector, in government, and in the cybersecurity product and service community.

So, we thought it would be interesting to feed the MITRE ATT&CK® Enterprise Matrix: https://attack.mitre.org/matrices/enterprise/ into Claude, which we used in the first use case here, and see how Atsign Platform: https://atsign.com/technology/atplatform/ stacked up against the attack matrix in the virtual eyes of Claude. Obviously, this is not a serious analysis, but interesting none the less.

Here are Claude's observations: I's an interactive dashboard mapping all 14 MITRE ATT&CK Enterprise tactics against Atsign Platform's capabilities. You can filter by coverage level and click cards to expand them.

Here's the strategic picture:

- **High Coverage (8 tactics)** — Atsign Platform *directly neutralizes* the attack vector:
- **Reconnaissance** — devices don't require open ports for communication, which blocks port scanning, a primary attack vector. No IPs, no DNS, no services to discover.
- **Initial Access** — NoPorts eliminates exploitable public-facing services and, since cryptographic identity replaces passwords, Valid Account abuse (default credentials, password spraying) becomes categorically impossible.
- **Credential Access** — All traffic is end-to-end encrypted and there are no passwords anywhere in the stack, so brute force, sniffing, MITM, Kerberoasting, and Pass-the-Hash all have nothing to operate on.
- **Discovery** — Zero open ports means nmap, service enumeration, and remote system discovery return nothing.
- **Lateral Movement** — only authorized parties can find and connect to devices, and every connection requires fresh cryptographic proof — compromising one node doesn't propagate.
- **Collection** — the Atsign Server stores only encrypted data owned by the Atsign, so even physical compromise of the server yields only ciphertext.
- **Command & Control** — No inbound ports means no inbound C2.
- **Medium/Partial Coverage (6 tactics)** — Atsign Platform removes the *network layer* of attacks (Persistence, Privilege Escalation, Execution, Defence Evasion, Exfiltration, Impact), but host-level techniques (process injection, scheduled tasks, ransomware via USB) require complementary endpoint security — which is where Atsign Platform is honest about its scope.

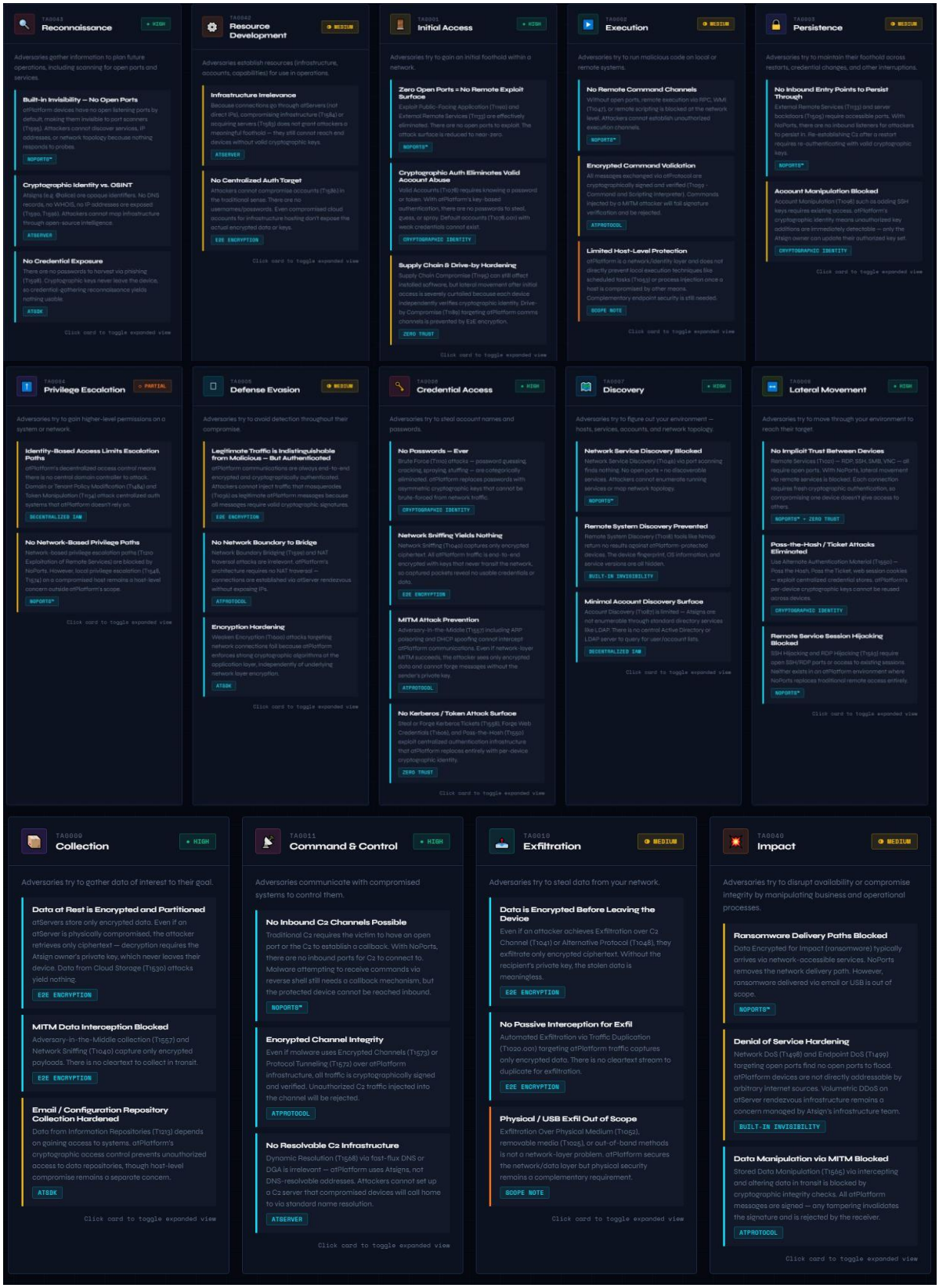

Figure 13 – Claude Analysis Of At Platform vs MITRE ATT&CK Enterprise Matrix

APPENDIX 2: USE CASE 2: PEMBROOK SECURE AI AGENT

Of the various forms of autonomous AI proliferating right now, agentic AI is the one being positioned at the heart of many company network infrastructures, whether purely internal or as part of a supply-chain type profile.

However, the common foundations for connecting agents both to each other and other tools have a fundamental architectural weakness, these being Anthropic' s Model Context Protocol (MCP), Google's Agent-to-Agent (A2A), and IBMs Agent Communication Protocol (ACP). That weakness is that they require trusted intermediaries to see the data in transit. API servers, cloud relays, and proxy gateways sit in the path of every message, creating obvious points of attack – each a single point of compromise. These three protocols are often described as complementary layers: MCP connects agents to tools, A2A connects agents to other agents, and ACP orchestrates the resulting workflows. Together they promise an interoperable ecosystem where specialised agents collaborate freely. But there is that very obvious security issue at large, as noted. If we look at a typical enterprise workflow, we see the following: a user's agent (via A2A) delegates a task to a specialist agent which invokes several MCP tools, one of which triggers a further A2A call to a third-party supplier's agent. At each hop, a new server operator gains visibility into the data. So, each operator adds to the aggregate attack surface, each and every time. It means, in attack terms, that a single compromised intermediary can exfiltrate data from every transaction passing through it. Not ideal…

Characteristic	MCP	A2A	atRPC (Pembrook)
Wire format	JSON-RPC 2.0	JSON-RPC 2.0	JSON-RPC 2.0
Transport encryption	TLS (HTTPS)	TLS (HTTPS)	TLS + E2E (AES-256 + RSA)
Server sees cleartext?	Yes	Yes	**No** (zero-knowledge relay)
Open inbound ports	Required	Required	**Zero**
Identity model	OAuth 2.0 / API keys	OpenAPI auth schemes	Cryptographic Atsign (PKAM)
Discovery	Manual config	Agent Cards (/.well-known/)	atDirectory (global, no data)

Figure 14 – Comparison of agentic protocol security properties

An alternative approach is to use the Atsign platform to create those agentic AI models – the basis of this use case. The challenge: to create a fully secure AI agent architecture while remaining fully compatible with the semantics of the aforementioned MCP, A2A, and ACP, while delivering end-to-end encryption, cryptographic identity, and zero open inbound ports to attack.

Use Case Overview

Pembrook was designed as an open-source, privacy-first AI assistant agent platform to demonstrate how to build a complete agentic system on the Atsign Platform. It aims to eliminate the entire class of much-documented network-exposure and supply-chain vulnerabilities that plagued OpenClaw, replacing them with cryptographic identity, E2E encryption, and skill sandboxing — all with zero open inbound ports on any component. The project implements JSON-RPC 2.0 message semantics, the same wire format used by MCP and A2A, but transports them over Atsign RPC instead of HTTPS. This substitution is the key insight: by changing only the transport layer while preserving the message format, Pembrook achieves full end-to-end encryption and zero-knowledge relay without sacrificing protocol compatibility. So, in the same way that the Atsign NoPorts application allows secure SSH and TCP connections without any open ports, so Pembrook applies this same proven model to MCP and agentic protocols.

The Underlying Architecture

The Pembrook system consists of three core Atsigns, each representing a distinct security domain. The @server Atsign represents a person and is used by the Flutter cross-platform application on all their devices. The @agent Atsign represents the AI daemon that runs in Docker and manages skills, memory, and audit logs. The @services Atsign is shared by all bridges (email, messaging), MCP server wrappers, and skill containers. Communication between these identities flows exclusively through the Atsign Platform's encrypted relay.

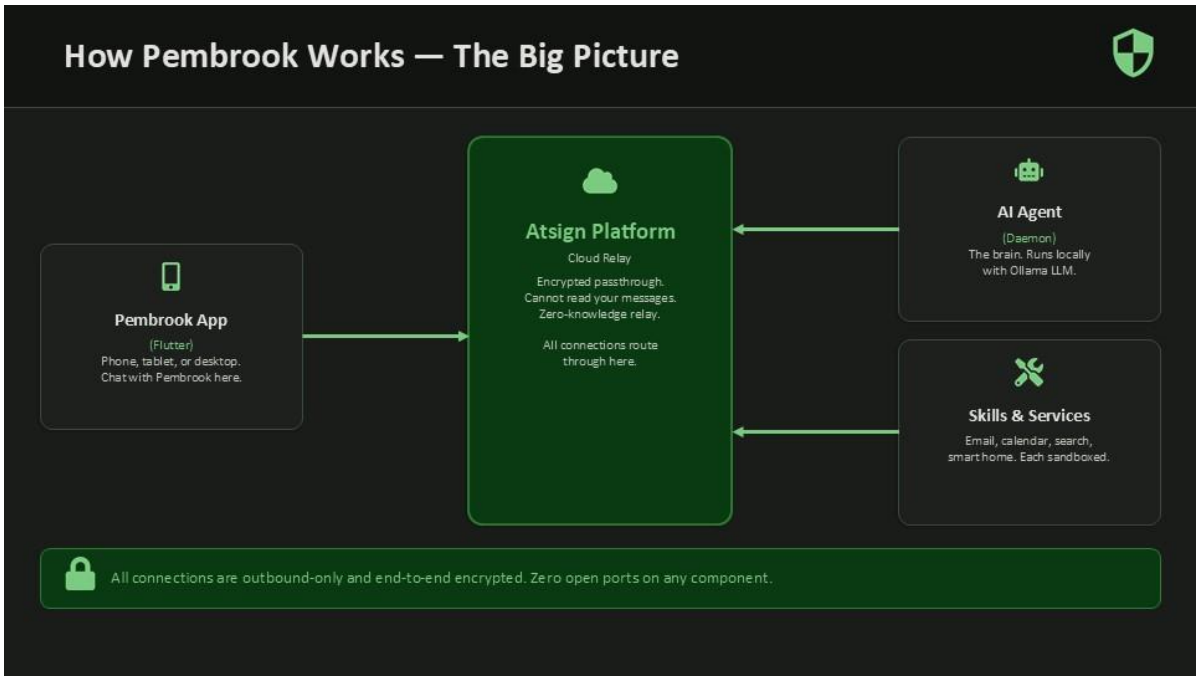

Figure 15 –The Pembrook architecture end-to-end

The agent's Gateway component implements an Atsign RPC server with a strict allowList containing only @owner and @services Every incoming request is verified against this list before any processing occurs.

When a message is sent through the Flutter app, the following sequence occurs:

- The app creates a JSON-RPC 2.0 request containing the command, conversation ID, and timestamp.
- The Atsign SDK on the device encrypts this payload using the shared symmetric key negotiated with @agent , wraps it in an Atsign Protocol notification, and transmits it outbound to the Atsign Platform relay.
- The relay stores the encrypted ciphertext on @agent 's Atsign Server.
- The agent daemon's Gateway, which maintains a persistent outbound connection to its Atsign Server, receives the notification, decrypts it locally using the agent's private key, and passes the cleartext JSON-RPC request to the Policy Engine.

Note: At no point in this chain does the Atsign Platform relay—or any other intermediary—see the message content.

The agent streams response tokens back to the user by batching approximately 80 characters per chunk and sending each as an Atsign RPC notification. All authenticated @owner devices receive these chunks in real-time via Atsign Platform broadcast, enabling multi-device synchronisation. A final sentinel message of done:true signals stream completion. This streaming model mirrors the Server-Sent Events pattern used by MCP and A2A, but with each event individually encrypted and authenticated.

Pembrook's skill system implements the tool-invocation pattern that MCP standardises, but adds multiple layers of isolation. Each skill runs in a disposable Docker container with strict resource limits (256MB memory, 0.5 CPU). Skills that do not require network access run with network isolation disabled entirely. The Policy Engine evaluates every tool invocation against configurable rules, and can escalate sensitive operations (such as sending emails) to human-in-the-loop (HITL) approval, where humans actively participate in the machine learning lifecycle. Audit entries for every skill invocation are stored as immutable Atsign Keys on the @owner's Atsign Server, meaning the agent cannot delete its own activity logs.

The crucial technical observation is that MCP, A2A, and ACP all use JSON-RPC 2.0 as their wire format. This means their message semantics are transport-agnostic by design. The JSON-RPC specification itself states that the protocol is transport-independent; the message format is the same whether carried over HTTP, WebSocket, stdio, or any other channel. This opens the door to running all three protocol semantics over Atsign RPC without modifying the protocols themselves.

An MCP server that provides tools (say, a database query tool) can be wrapped so that its JSON-RPC 2.0 requests and responses are transported via Atsign RPC instead of HTTP, as demonstrated by the Pembrook project. The benefits are immediate: the MCP server infrastructure operator never sees the queries or results.

Even Atsign's own infrastructure, which relays the messages, sees only encrypted ciphertext. This directly addresses the vulnerability identified by Atsign's own analysis that MCP's current architecture is scaling a broken trust model.

A2A's agent-to-agent messaging translates naturally to the Atsign RPC model. In standard A2A, a client agent discovers a remote agent via its Agent Card, then sends task messages over HTTPS. In the Atsign RPC variant, agent discovery uses the Atsign Directory (which maps Atsigns to Atsign Servers) combined with public metadata stored on each agent's Atsign Server—functionally equivalent to an Agent Card.

Task messages (message/send, message/stream) are JSON-RPC 2.0 payloads transported over Atsign RPC. Because Atsign RPC supports both request-response and notification patterns, it accommodates A2A's synchronous, streaming, and asynchronous push models. The critical difference is that the agent platform hosting the remote agent no longer needs to be trusted with the content

ACP's workflow orchestration semantics—task delegation, stateful sessions, and progress tracking—map to the Pembrook Orchestrator's existing capabilities. The orchestration layer never needs to expose an HTTP endpoint, thereby maintaining the zero-port-exposure posture.

The Creation Process - Components

As already noted, the basic elements of the AI Agent here are defined as the Three-Identity Model - @owner, @agent and @services.

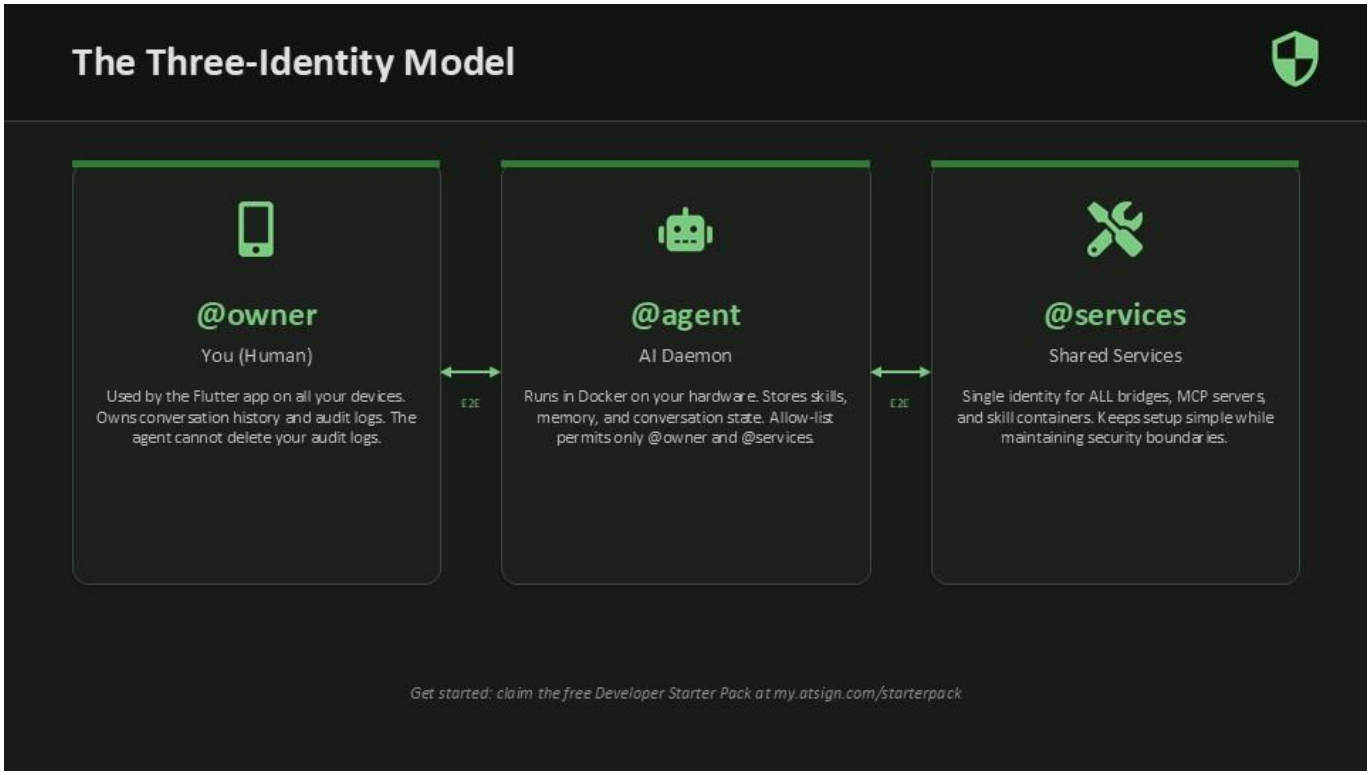

Figure 16 –The Three-Identity Model

@owner (You (Human)): Used by the Flutter app on all your devices. Owns conversation history and audit logs. The agent cannot delete your audit logs.

@agent (AI Daemon): Runs in Docker on your hardware. Stores skills, memory, and conversation state. Allow-list permits only @owner and @services.

@services (Shared Services): Single identity for ALL bridges, MCP servers, and skill containers. Keeps setup simple while maintaining security boundaries.

So what happens when you actually chat to the Ai model? The message flow is as follows:

1. You type a message in the Flutter app.
2. App encrypts it E2E, sends via Atsign Platform.
3. Gateway checks cryptographic identity.
4. Policy Engine: allowed? Need approval?
5. Orchestrator sends to local Ollama LLM.
6. If tools needed → sandboxed skill runs.
7. Response streams to ALL your devices.
8. Immutable audit log written to @owner.

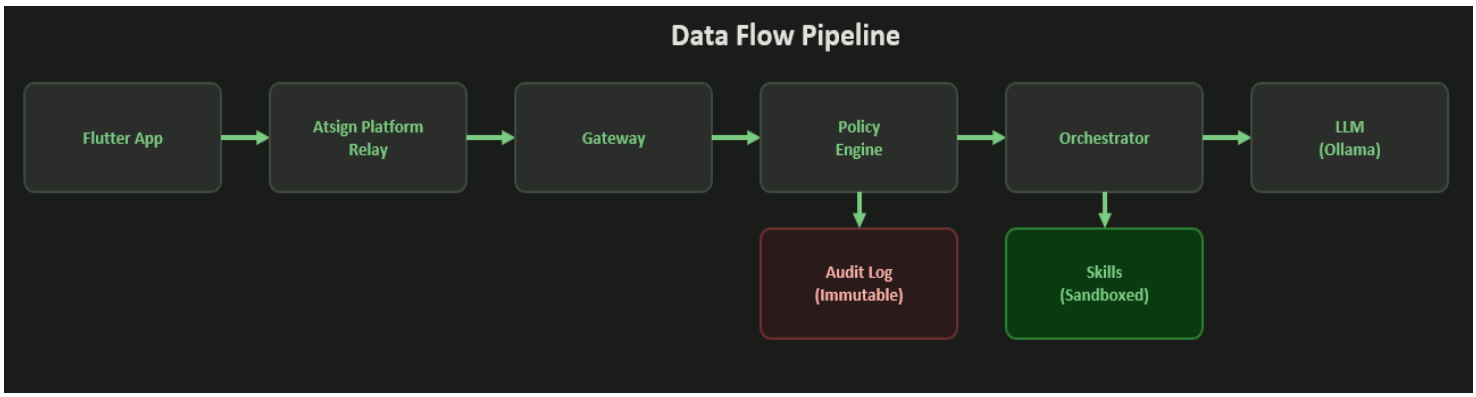

Figure 17 –The Data Flow Pipeline

Inside the Agent Daemon (@agent) the data flow moves from the Gateway (Atsign RPC server verifies identity via allow-list) to the Policy Engine (Allow/Deny/Escalate to HITL YAML-based rules) then to the Orchestrator (Routes tasks & manages tool chains Up to 10 iterations).

Orchestration is underpinned by the LLM Router (Local: Ollama (private) External: sanitized queries only), the Memory Service (Conversation history User preferences & profile) and the Audit Service (Immutable logs on @owner Agent cannot delete).

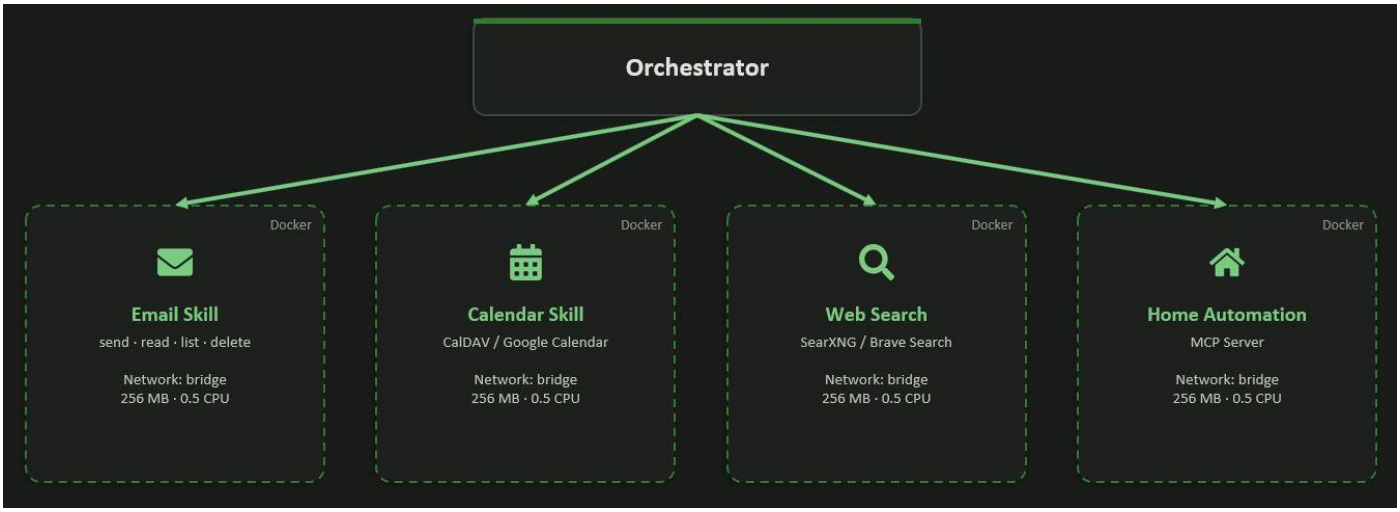

Figure 18 –Skill sandboxing: Secure Plug-In Architecture

The Orchestrator utilises skill sandboxing as a secure plug-in architecture in order to ensure that each skill runs in isolation, that non-network skills get-network=none and that containers are destroyed after execution.

Atsign AI Architect was used to design the blueprint for the app, defining actors, dataflows and permissions in a visual fashion – no programming required. It then exports a precise JSON specification as an LLM prompt. Note this is a rigid contract, not a vague prompt that could be misinterpreted. Any LLM can be used to then build the app.

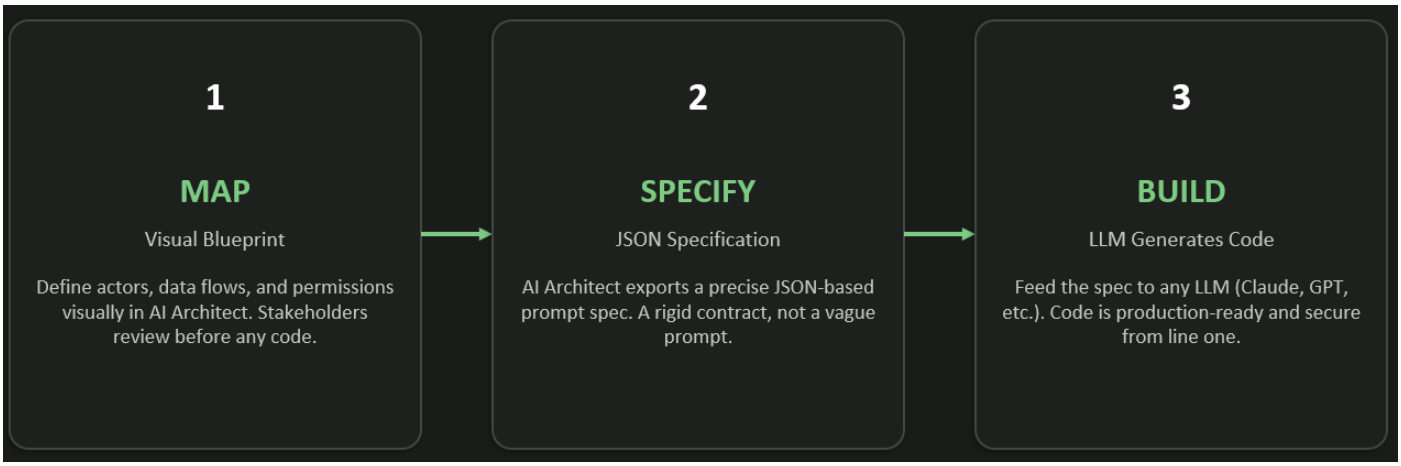

Figure 19 – AI Architect

The Finished Application

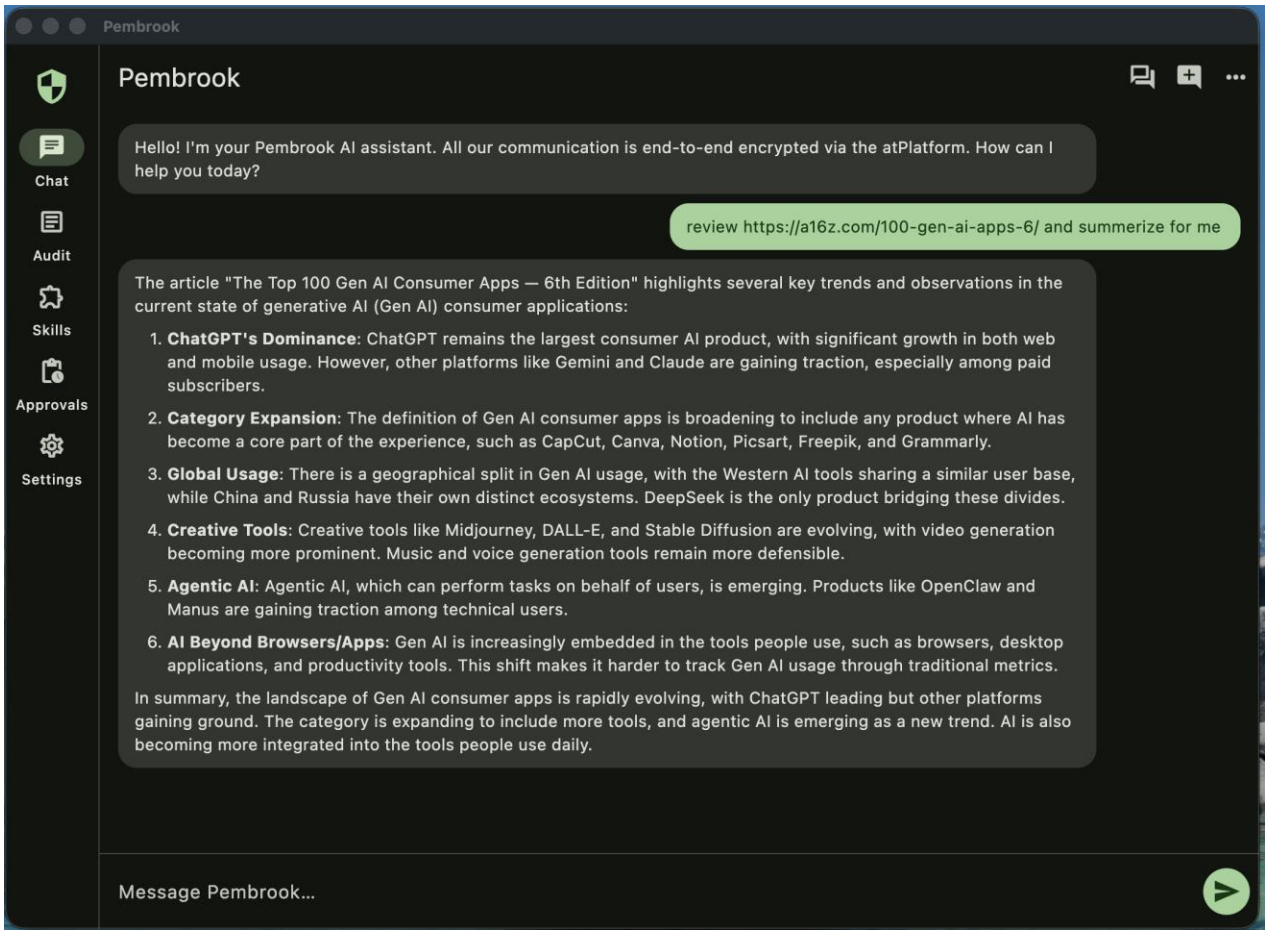

Figure 20 – The Pembrook AI Assistant

In use, Pembrook (the name is inspired by a classic "English butler type" as observed in many films) is a typical AI assistant with the classic look and feel.

Key Features

- **Real-time Progress Indicators:** see live status updates during multi-step tasks (e.g., "🌐 Fetching content from BBC...", "✉ Sending email...").
- **Smart Timeout Management**: 90-second timeout resets automatically with each progress update or content chunk.
- **Multi-Device Sync**: conversations sync across all your devices in real-time via encrypted Atsign Keys.
- **Tool Call Streaming**: watch the agent's reasoning and tool invocations as they happen.
- **Urgency-Based Notifications**: agent can send immediate alerts (critical/high) or queue low-priority updates for daily digest.

- **MCP Integration**: built-in browser automation, with extensibility for home control, databases, and more.
- **Encrypted Audit Logs:** immutable audit trail stored on your Atsign Server, viewable in the app.
- **Live Log Viewer:** web-based development tool at http://localhost:9090 with service filtering and keyword highlighting.
- **Zero Trust Architecture:** no open ports, cryptographic authentication, E2E encryption on all channels.

In terms of how the security stacks up compared with a classic OpenClaw type development, here are the key differences:

Threat	OpenClaw	Pembrook
Network exposure	Port 18789 open to internet (30,000+ exposed)	**Zero open ports** — all outbound-only
Authentication	None by default	Cryptographic atSign PKAM verification
Credentials	Plaintext `.env` files	Encrypted AtKeys, never plaintext
Skills/plugins	ClawHub (386 malicious packages in days)	Verified developer atSign + Docker sandbox
Prompt injection	Full agent authority	Untrusted input tagged, restricted context
Memory poisoning	All inputs equal authority	Source-tagged trust levels, non-escalatable
Confused deputy	Agent tricked into misusing tools	Context-aware tool filter per task type
Supply chain	Anonymous publishing	Developer identity required, signed packages
Lateral movement	Full home directory access	Pico-segmented per-skill / per-tool
Audit	None	Immutable AtKeys on owner's atServer

Figure 21 –Pembrook vs OpenClaw: Security Differences

Security Overview: What Has Actually Changed?

From an overall security standpoint, in terms of what potential threats still exist and which don't, it is important to understand the "bigger picture".

Moving agentic protocols onto Atsign RPC addresses a specific and well-defined set of threats, but we need to be precise about what it does and does not change.

Threats Eliminated

Data exfiltration by infrastructure operators

In the standard model, every MCP server, A2A agent host, and API gateway can read the cleartext data flowing through it. With Atsign RPC, these intermediaries see only ciphertext. Even a compromised relay yields no usable data

Network exposure and port scanning

Standard deployments require open inbound ports for HTTP/HTTPS servers. Each open port is a potential entry point. Pembrook's zero-port architecture eliminates this entire attack surface. An nmap scan of a Pembrook deployment returns zero open ports.

Credential theft and replay

OAuth tokens and API keys can be stolen and reused. Atsign authentication requires cryptographic proof of key possession, which cannot be replayed from a different device.

Man-in-the-middle attacks

Even with TLS, a compromised certificate authority or a misconfigured proxy can intercept traffic. The Atsign Protocol's end-to-end encryption is independent of the transport layer's TLS, providing defence in depth.

Threats That Remain

Endpoint compromise

If the device running the agent is compromised, the attacker gains access to the private keys and can decrypt all messages. Endpoint security remains essential.

Prompt injection and reasoning attacks

The encryption of the transport does not protect against malicious content within the messages. Pembrook addresses this separately through prompt tagging and trust levels, but the Atsign Platform itself does not solve this class of problem.

LLM provider visibility

When the agent routes queries to an external LLM (via the LLM Router's sanitisation pipeline), the external provider sees the sanitised query. Fully private inference requires local models, which Pembrook supports via Ollama.

What follows is an overview of relative architecture properties of the standard (HTTPS) model versus at Atsign RPC-based approach of Pembrook:

Architectural Property	Standard (MCP/A2A/ACP over HTTPS)	atRPC-Based (Pembrook)
Message format	JSON-RPC 2.0	JSON-RPC 2.0 (identical)
Transport encryption	TLS at transport layer	TLS + AES-256/RSA E2E at application layer
Relay/server sees cleartext	Yes — full visibility	No — zero-knowledge relay
Identity model	Tokens, API keys, OAuth	Cryptographic key pairs per Atsign
Open inbound ports required	Yes (HTTP/HTTPS)	Zero (all outbound)
Agent discovery	Agent Cards over HTTPS, manual config	atDirectory + public atServer metadata
Credential theft risk	Token/key replay possible	PKAM requires key possession proof
Audit trail integrity	Server-controlled logs	Immutable AtKeys on owner's atServer
Multi-device sync	Application-level sync required	Built-in via Atsign Platform self-key propagation

Table 2: Architectural comparison between standard HTTPS-based and atRPC-based agentic systems.

Figure 22 –Pembrook vs Standard (HTTPS) Architectures

Other Benefits Of The Pembrook Architecture

Zero-configuration networking

One of the most significant practical benefits is the elimination of network configuration as a deployment concern. Standard MCP and A2A deployments require firewall rules, security group configurations, DNS records, SSL certificate management, and often VPN tunnels.

A Pembrook-style deployment requires none of these. The docker-compose file starts the agent, Ollama (for local LLM inference), and any MCP server wrappers, and they begin communicating immediately via the Atsign Platform. This reduces deployment from a multi-team, multi-week infrastructure project to a single-developer, single-day task.

Compliance and data sovereignty

Note: For organisations operating under GDPR, HIPAA, or other data protection regulations, the zero-knowledge relay architecture provides a powerful compliance posture.

Since the Atsign Platform relay cannot access the data it transports, it does not need to be classified as a data processor under GDPR. The data controller (the organisation running the agent) retains full control of the encryption keys and thus full control of the data. This simplifies compliance architectures by eliminating the need to audit and contract with every intermediary in the communication chain.

Cross-organisational agent collaboration

The most compelling use case for A2A-over-atRPC emerges when agents need to collaborate across organisational boundaries. In the standard model, both organisations must trust every intermediary platform. With Atsign RPC, Organisation A's agent and Organisation B's agent negotiate a shared encryption key via the Atsign Protocol, and their subsequent task messages are opaque to both organisations' hosting infrastructure. Each organisation maintains its own Atsign-based access policies. This enables secure supply-chain automation, cross-company workflow orchestration, and multiparty AI collaboration without any party needing to trust the other's infrastructure.

PEMBROOK: CONCLUSIONS

The agentic AI protocols that are being adopted today: MCP, A2A, and ACP - solve real and important problems around tool integration, agent discovery, and workflow orchestration. But they share a fundamental architectural assumption that intermediary servers can be trusted with cleartext data. This assumption is increasingly untenable as agents handle sensitive enterprise data across organisational boundaries.

The Atsign Platform, through the Atsign Protocol and Atsign RPC mechanism, provides a proven, production-ready alternative transport that delivers end-to-end encryption, cryptographic identity, and zero network exposure. Because MCP, A2A, and ACP all use JSON-RPC 2.0 as their message format, layering their semantics over Atsign RPC requires no modification to the protocols themselves, only a change in the underlying transport.

The Pembrook project demonstrates that this is not merely theoretical. A fully functional, privacy-first agentic system—with streaming chat, multidevice sync, skill sandboxing, tool invocation, memory, audit, policy enforcement, and human-in-the-loop approval—can be built entirely on this architecture. The code is open source (BSD 3-Clause) and available at **github.com/pembrookai/Pembrook**.

The question for the agentic AI community is not whether to adopt MCP, A2A, or ACP—these protocols are already achieving critical mass. The question is whether the transport layer beneath them can be upgraded to match the sensitivity of the data they carry. The Atsign Platform offers one compelling answer: keep the protocols, change the plumbing, and make the intermediaries blind.