Technical Exchange — TSM-FR

VPM Integration: Questions for TrellisWare

February 11, 2026

Guillaume Draznieks — Procomm-MMC

Where we stand

✓ VPM crypto module implemented and tested — all Part 3/4/5 test vectors pass (KDF, Voice, Data/PLI)

✓ Full TCVI Extended Protobuf interface implemented (TcviTransport, GetChannel, Transform, all 22 Bypass messages)

✗ Transport layer — how to carry TCVI messages to/from the radio — is not covered by the SDD

The SDD defines the cryptographic operations and the Protobuf message format. Our implementation covers both completely. What remains is the transport: how a physical VPM module connects to and exchanges TCVI messages with the radio. This is radio-firmware-specific and only TrellisWare can provide it.

Questions — Overview

#	TOPIC	PRIORITY	KEY CONCEPTS
1	Network transport	Critical	ports, protocol, framing, API definition
2	Data flow architecture	Critical	Red/Black path, firmware routing, module placement
3	External VPM support	Critical	firmware readiness, TCVI exposure, timeline
4	Connection lifecycle	Important	init sequence, handshake, fallback, recovery
5	Traffic enforcement	Important	plaintext leak, mandatory routing, demo integrity
6	Integration documentation	Helpful	sequence diagrams, reference code, app notes
7	Virtual dev environment	Helpful	simulator, SIL, test harness

Detail

1. Network transport for TCVI Extended messages

The SDD defines TcviTransport as a set of Protobuf3 messages (GetChannel, GetChannelResponse, Transform, Bypass) but the .proto file contains no service definitions — no gRPC stubs, no RPC methods — which tells us transport is implementation-specific rather than gRPC-based. Our module will connect to the radio over the existing USB-Ethernet interface (10.1.0.2). We have mapped the management API on port 443 (MQTT over WebSocket, TLS 1.3, client certificate auth, 125 topics) and confirmed it carries no TCVI-related traffic. We need to know which network endpoint on the radio carries TCVI Extended messages so we can build our transport layer.

- Which TCP/UDP port(s) on the radio carry TCVI Extended messages?
- What application-layer protocol? (raw TCP, WebSocket, MQTT on a different broker, custom framing, other?)
- What message framing wraps individual Protobuf messages? (varint-delimited per Protobuf convention, fixed-length header, other?)
- Is there an API definition available (OpenAPI, .proto with service defs, interface spec document)?

2. Data flow architecture — where does the external VPM module sit?

The SDD states that GetChannelResponse is sent "to both Red and Black services," and describes the TCVI interface as sitting between a radio and a VPM module that can be "logical or physical." This implies the crypto module has a dual-facing role — intercepting traffic between the application layer (voice codec, data apps, PLI engine) and the RF transmission layer. For an internal (logical) module this is straightforward: it's a shim in the firmware's data path. For an external (physical) module like ours, we struggle to understand the exact data flow. We need a clear picture of how our module inserts itself: which radio-internal services talk to us, through which APIs, and how encrypted traffic is handed back to the radio for RF transmission.

- Can you describe the end-to-end data flow: Red-side app → [what API?] → external VPM module → [what API?] → Black-side RF?
- Does the radio's firmware actively route traffic through the external module, or does the module intercept independently?
- Conceptually, where in the radio's software architecture does the external TCVI boundary sit?

3. External (physical) vs internal (logical) VPM module

The SDD describes the TCVI interface as sitting between a radio and a VPM module that can be "logical or physical." We are implementing a physical external module — a separate hardware device connected to the radio over the network. This means the radio firmware must expose the TCVI interface as a network-accessible service, rather than keeping it as an internal function call boundary. We want to understand whether this external exposure already exists in the current firmware, or whether it requires a specific firmware build or update. This is time-critical because if a firmware update is required, we need to factor in its delivery timeline before we can begin integration testing.

- Does the current radio firmware expose the TCVI interface to external network clients, or is it internal-only today?
- If not currently exposed: is a firmware update required? What is the expected availability?
- Is there a configuration flag or preset parameter to enable external TCVI?

4. Connection lifecycle and initialization

Once we know the transport (Q1), we need to understand the connection lifecycle. The SDD defines the message flow (GetChannel → GetChannelResponse → Transform/Bypass) but not who initiates the connection, whether there is a registration or handshake step before TCVI messages can flow, or what happens when the connection drops. These are firmware-defined behaviors that determine how our module bootstraps itself on power-up and how it recovers from failures. Understanding the lifecycle is essential for implementing a robust, field-reliable module that can survive disconnections and power cycles gracefully.

- Who initiates the TCVI connection — the radio or the external module?
- Is there a discovery/registration step before GetChannel messages can be exchanged?
- When connected, does the radio route ALL voice/data/PLI through the module, or only preset-selected channels?
- On disconnection: does the radio fall back to unencrypted, or does it stop transmitting?

5. Traffic enforcement — guaranteeing all data flows through the VPM module

For the VPM to provide meaningful security, we need assurance that ALL voice, data, and PLI traffic is routed through our crypto module before transmission — and that no plaintext can leak around it. If the radio can still transmit unencrypted traffic independently of the module (for example during initialization, on fallback, or on channels not covered by a GetChannel mapping), the security guarantee breaks down. This is critical for the demo scenario: a third radio without the module must not be able to understand ANY traffic, which means the radio must enforce that nothing goes out unencrypted when a VPM module is active.

- When a VPM module is connected, does the radio enforce that ALL traffic passes through it before RF transmission?
- Can plaintext traffic bypass the module on any channel or in any mode (e.g. during startup, on unconfigured channels)?
- Is there a radio-side mechanism to block unencrypted transmission entirely when VPM is active?

6. Additional integration documentation

We built the entire crypto module from the SDD alone — all six parts — and we are fully autonomous on the cryptographic implementation. What we lack is documentation about the integration layer: how to connect our module to the radio, what the expected message sequences look like in practice, and what edge cases to handle. We fully respect the boundaries around what can and cannot be disclosed given export control constraints. If there are additional integration references within the framework of our partnership — even partial, even informal — they would significantly reduce our iteration time.

- Transport-level integration guide, application notes, or sequence diagrams for external TCVI?
- Reference implementation or test harness? If specific NDA or export framework is needed, happy to discuss.

7. Virtual development environment

For iterative development and testing, constant physical radio access creates a bottleneck — both logistically and in terms of development speed. Ideally, we would like to develop and test our TCVI integration against a virtualized or simulated radio environment that exposes the same TCVI interface as the real hardware. This would allow us to run automated tests, iterate on the transport layer, and validate message flows without needing a radio on the bench for every development cycle. If TrellisWare has a software-in-the-loop environment, radio simulator, or even a TCVI stub server, access to it would dramatically accelerate our integration timeline.

- Do you have a radio simulator, emulator, or software-in-the-loop environment that exposes the TCVI interface?
- If so, could we get access or guidance to set one up for our development workflow?

Our integration planning follows the milestone schedule below. Answers to the questions above will allow us to finalize each phase.

OUR MILESTONE	TARGET	DEPENDS ON
Transport layer implemented	End of February	Q1 + Q3 + Q4
Hardware prototype boots	Mid-April	Q1 + Q2 (interface type)
End-to-end test with real radio	May	Q1 through Q5 + Q7
Demo-ready	June 1	All of the above