

● WHITEPAPER · LOCAL SYNC AND DATA INTEGRITY

Device-to-app sync for VAHA: architecture, and the integrity and security of the current build

AUTHORS

Ritesh Bonthalakoti

REVIEWER

Bharat Meda

VERSION

v1.0

DATE

2026-10-22

STATUS

Published

DISTRIBUTION

Public

CONTENTS

Abstract	6 Analysis and discussion
1 Introduction and objective	7 Limitations and threats to validity
2 Requirements and acceptance criteria	8 TRL assessment
3 System under test	9 Conclusions and next steps
4 Method	10 References
5 Results	Appendices

ABSTRACT

Abstract

VAHA keeps voice notes off the cloud by moving each capture from the device to its companion app over the local Wi-Fi network. This paper describes that sync path and checks the current build, as of 2026-08-17, against its design: by reading the code, by inspecting the released app (v1.1.0), and by running the device server's code on a development machine. The app's sync has not yet been tested on a handset. The build falls short in three places. The app deletes the device's copy as soon as a download returns HTTP 200, without comparing the MD5 checksum the device provides, so a damaged transfer would leave no good copy anywhere. The app and the device share one compiled-in access token. All traffic, including audio, transcripts and that token, travels as plain HTTP. Of four requirements taken from the product README, one is met, one partly met and two not met. The paper shows how each gap can be closed, including TLS to a device that has no public certificate, and ends with a checklist for local-only sync that applies beyond VAHA. The sync path is at technology readiness level (TRL) 3, an experimental proof of concept.

KEY POINT

Key findings

- **Integrity:** the app deletes the device copy after HTTP 200 without checking the MD5, and the released app contains this code. A damaged download becomes the only copy.
- **Access:** the app and the device share one default token, `vaha-dev-2026`. Run on a test machine, the server let a client holding it list, download and delete captures; with its check switched off, it let anyone.
- **Transport:** plain HTTP. The released app allows it and the server does not accept TLS. Audio, transcripts and the token can be read by any device on the same network.
- **Readiness:** TRL 3. The README's performance figures have no recorded method. Next: restore the MD5 check and test it with deliberately corrupted transfers.

SECTION 1

Introduction and objective

Every capture VAHA records reaches the user through one path: the companion app downloads it from the device over the local network, stores it, and tells the device to delete its copy. After that the handset holds the only copy. The product's privacy position, that no audio or text leaves the home, also rests on this link. So the sync path has to deliver each recording intact and keep it away from anyone else on the network.

The problem is common to local-first devices: there is no cloud service to hold a second copy, no public certificate authority for a device on a home network, and a phone operating system that blocks plain HTTP by default. The ways of closing the gaps in section 6.4 and the checklist in section 9 apply to any device that syncs this way.

This paper answers four questions:

- 1. Does the sync path keep captures on the local network?
- 2. Is each capture verified before the device deletes its copy?
- 3. Who can read captures from the device?
- 4. Are captures and credentials protected on the network?

In scope: the sync protocol between the device's HTTP server and the app; authentication on both sides; integrity checking; and the network configuration of both sides.

Out of scope: the app's user interface; the optional Groq transcription fallback and Notion export, which are separate features; storage of third-party API keys; the WebSocket telemetry channel; and penetration testing.

SECTION 2

Requirements and acceptance criteria

The requirements come from the product README (reference 1). They were written down for this paper, after the code, so they state the design the build is checked against, not acceptance criteria set before testing (section 8).

ID	Requirement	Criterion	Basis
R1	Local-only sync	Captures move from device to app over the local network, with no cloud service in the sync path	README: "No audio or text ever touches the cloud"
R2	Verified before purge	The app compares each capture's MD5 with the device's value and deletes the device copy only when they match	README: the app "verifies the MD5 checksums" before purging; checksum verification "confirmed on every capture package ... with zero corrupted transfers"
R3	Access limited to the paired app	The device serves captures only to a client holding a credential unique to that device	Implied by the privacy position

ID	Requirement	Criterion	Basis
R4	Protected in transit	Audio, transcripts and credentials cannot be read or altered by another device on the same network	Implied by the privacy position

SECTION 3

System under test

3.1 The sync path

On the device, the capture pipeline writes each recording as a package in `captures/YYYY/MM/DD/<id>/: audio.wav , transcript.json , metadata.json and checksum.md5` , an MD5 checksum (a 128-bit fingerprint) of the audio. A FastAPI server on port 8080 serves the packages under `/api/v1/captures` . It checks a bearer token, a shared secret sent with every request, on each capture request. On the handset, `SyncService` runs the sync and keeps its state in two SQLite tables: `captures` , with a sync state per capture, and `sync_queue` , with one job per pending download. Figure 1 and Table 3.1 follow one capture through.

ONE CAPTURE, FROM DEVICE TO HANDSET

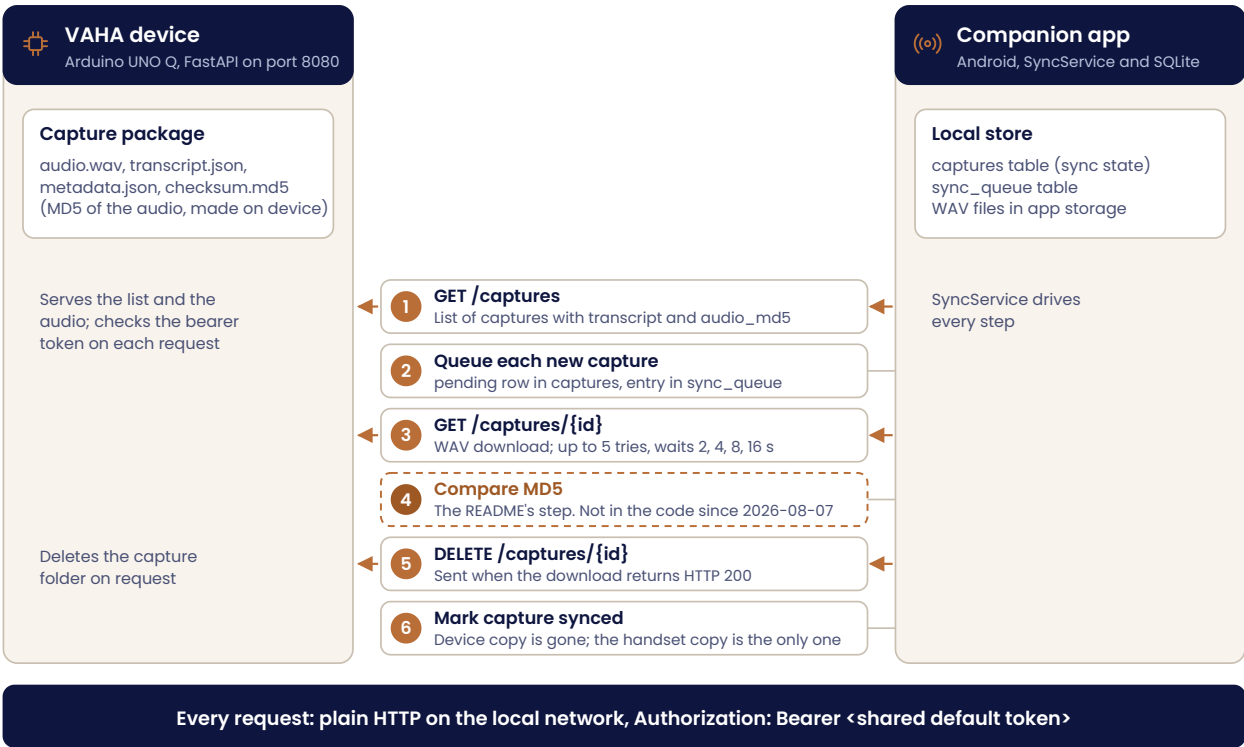


Figure 1 One capture from device to handset, as the code runs it. Paths are under `/api/v1` . Step 4 is the MD5 comparison the README describes; the app does not perform it.

Table 3.1: Sync steps as implemented (source locations in Appendix A)

Step	What happens	Source
1	The app requests the capture list. Each entry carries the transcript, the audio size and <code>audio_md5</code>	<code>SyncService.ts</code> line 59; <code>captures.py</code> line 80

Step	What happens	Source
2	Each new capture with a non-empty transcript gets a pending row in captures and a capture_download job in sync_queue	SyncService.ts lines 79–120
3	The app downloads the WAV file straight to app storage	SyncService.ts lines 336–357; captures.py line 85
4	If the download fails or returns anything but HTTP 200, the app retries: up to 5 attempts, waiting 2, 4, 8 and 16 s (calculated) plus up to 0.5 s of random jitter each time, then marks the capture failed	SyncService.ts lines 276–313 and 399–403; ApiCompatibility.ts lines 43–49
5	When the download returns HTTP 200, the app sends a delete request and the device removes the capture's whole folder	SyncService.ts lines 368–375; captures.py line 93
6	The app marks the capture synced . A failed delete is only logged	SyncService.ts lines 190 and 202

With 5 attempts the 60 s cap on the wait is never reached; a capture is given up after about 30 s of waiting (calculated).

3.2 Technology stack

Layer	Technology
Device	Arduino UNO Q (ABX00087), Linux side
Device server	Python, FastAPI, run by Uvicorn on port 8080, all network interfaces
Companion app	Expo SDK 54, React Native, Android build
App storage	SQLite through Drizzle ORM; WAV files in app storage
Transport	HTTP on the local network, bearer-token authentication
Device telemetry	WebSocket on the same port (out of scope)

3.3 Companion app

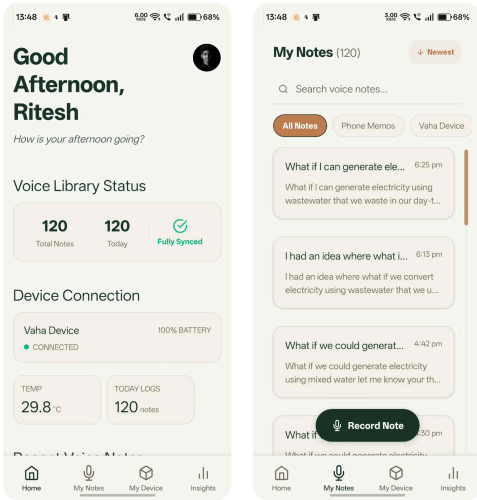


Figure 2 Companion app home view and notes view. The home view shows 120 notes recorded that day, all reported as synced; each came through the sync path in Figure 1. The session itself has no recorded date; the images entered the repository on 2026-08-17 (reference 1), so it ran on or before that day (section 6.2).

● SECTION 4

Method

4.1 What was examined

Item	Details
Source	VAHA source repository (reference 1), as of the last change to its code on 2026-08-17
Device server	python/server : configuration, authentication, capture routes and capture service
Companion app	mobile/ : sync service, device transport, configuration and Android manifest settings (app.json)
Device	Arduino UNO Q (ABX00087)
Released app	vaha-companion-v1.1.0-arm64-v8a.apk and -armeabi-v7a.apk from the repository's release folder; SHA-256 in the raw data. The APK attached to the repository's v1.1.0 release page is a separate build and is not covered here
Test machines	Release inspection and access test (steps 6 and 7): development computer, macOS on Apple silicon, Python 3.13.3, Android SDK build-tools 36.0.0 for reading the app manifest. Loopback timing (step 8): development computer, Windows, Python 3.12.10. 120-capture run (step 8): development computer, Windows on AMD64, Python 3.12.10. Server packages as in analysis/requirements.txt
Handset (Figure 2 only)	Android handset; model and OS version not recorded

4.2 Procedure

The scripts and data for every measurement below are published with this paper (reference 4). They sit in the analysis/ and data/ folders of the VAHA repository, and paths such as analysis/server_access_test.py below are relative to its root. The server scripts run the device server code from the same checkout. They were added on 2026-10-22 without changing that code, so a checkout from that date repeats the runs reported here; a later checkout tests whatever the code has become.

1. Follow one capture through the code, from the device's capture package to the app's local store and the delete request.
2. For authentication, find where each side gets its token, where the device checks it, and how the check can be switched off.
3. For the network, read the app's cleartext setting and the server's address, port and TLS configuration.
4. Use the repository history to date when each behaviour was introduced (Appendix B). Where the history was rewritten or folders were moved, take the date of the commit that first contains the code, not the date a file appears at its current path.
5. Compare each step with the product README's description of sync.
6. Inspect the released app: read its manifest, and search its JavaScript bundle for text that only the code before or after 2026-08-07 contains (analysis/inspect_release_apk.py).

7. Run the device server's code unchanged on the test machine, bound to the machine itself, with three synthetic captures written by the server's own capture code (one second of a 440 Hz tone each). Send list, download and delete requests with no token, a wrong token and the default token, first with the default settings and then with `VAHA_AUTH_ENABLED=false`, and try a TLS handshake on the server's port (`analysis/server_access_test.py`).
8. On the same unchanged server code on loopback: time 10 downloads of an 8 s synthetic capture (`analysis/sync_performance_test.py`), and run 120 synthetic captures of 2–14 s through list, download and delete with the default token, recording the HTTP status of each and whether each downloaded body matches the server's `audio_md5` (`analysis/dated_120_capture_session.py`). Both are summarised by `analysis/summarise_sync_runs.py` .

Not yet done: the app's sync on a handset, packet capture on the network, corrupted transfers, and timing on the device over Wi-Fi.

SECTION 5

Results

5.1 Known gaps in the current build

The current build falls short of the design in section 2 in three places. Each gap is given with where it sits in the code, how it came about and what it allows.

CAUTION

Gap 1: the MD5 is not checked before the device copy is deleted

The device writes an MD5 of every recording and sends it in the capture list as `audio_md5` . From 2026-07-22, when the sync service was first written, it compared it with an MD5 of the downloaded file, but the comparison failed open: if the MD5 could not be computed, or the check raised an error, the capture was treated as good. On 2026-08-07 the download was rewritten to use `expo-file-system`'s `downloadAsync` , which writes straight to a file, to fix out-of-memory failures on large WAV files. The comparison went out with the old download code and has not been restored. Since then the only test before the delete request is that the download returned HTTP 200 (`SyncService.ts` lines 359–375). A placeholder for the check, `HttpDeviceTransport.verifyChecksum()` , always returns `true` and is not called anywhere.

```

mobile/src/features/devices/transport/HttpDeviceTransport.ts
447  /**
448   * Exposed for internal use by DeviceSyncService checksum verification.
449   * Validates a received ArrayBuffer against an expected MD5 base64 string.
450   *
451   * NOTE: React Native does not ship a native MD5 implementation.
452   * This is a placeholder that always returns true in Phase D; the real
453   * implementation will use expo-crypto in a follow-up once the transfer
454   * pipeline is validated end-to-end.
455   *
456   * @see https://docs.expo.dev/versions/latest/sdk/crypto/
457   */
458   static verifyChecksum(
459     _buffer: ArrayBuffer,
460     _expectedMd5Base64: string,
461   ): boolean {
462     // TODO(ADR-0014): Replace with expo-crypto MD5 implementation before production.
463     // Issue: vaha/issues/XXX
464     return true;
465   }

```

Figure 3 `HttpDeviceTransport.ts`, lines 447–465. The `verifyChecksum()` placeholder returns `true` without reading its inputs. Nothing in the app calls it; the comparison the sync path did use was removed on 2026-08-07.

CAUTION

Gap 2: one default token for every device and app

The app compiles in `vaha-dev-2026` as its default device token (`AppConfig.ts` line 52), used whenever no token is set at build time (`ApiCompatibility.ts` line 58). The device server has the same default (`config.py` line 22) and checks it on every capture, sensor, settings, storage and device-status request (`auth.py` line 52). Setting `VAHA_AUTH_ENABLED` to `false` switches the check off entirely (`config.py` line 25). The shared token is a development-stage stand-in. The design replaces it with a session token derived from a secret exchanged over Bluetooth LE at pairing, which is not built yet (`config.py` lines 19–21; `auth.py` lines 9–10).

```
mobile/src/core/config/AppConfig.ts
41 export const appConfig: AppConfig = Object.freeze({
42   env: parsedEnv.NODE_ENV,
43   database: Object.freeze({
44     name: parsedEnv.DATABASE_NAME,
45   }),
46   storage: Object.freeze({
47     keystoreAlias: parsedEnv.SECURE_KEYSTORE_ALIAS,
48   }),
49   useMockDevice: parsedEnv.USE MOCK_DEVICE,
50   defaultDeviceIp: "192.168.88.28",
51   defaultDevicePort: 8080,
52   defaultDeviceToken: "vaha-dev-2026",
53 });
```

Figure 4 `AppConfig.ts`, lines 41–53. The default device token at line 52 is the same string in every build that does not set its own.

CAUTION

Gap 3: plain HTTP, enabled for local hardware

Android blocks plain HTTP by default. `usesCleartextTraffic` was enabled in `app.json` (line 24) on 2026-08-10, "for local HTTP hardware communication", so the app could reach the device's server. The server listens on all network interfaces (`config.py` line 14) and is started without TLS (`app.py` line 33). Every request and response, including the bearer token, the transcripts in the capture list and the audio, crosses the network unencrypted.

```
mobile/app.json
23 "package": "com.vaha.companion",
24 "usesCleartextTraffic": true,
25 "permissions": [
26   "android.permission.BLUETOOTH",
```

Figure 5 `app.json`, lines 23–26. `usesCleartextTraffic: true` at line 24 lets the Android app send and receive unencrypted HTTP.

Table 5.1: Gaps summary

Gap	Where	Since	Consequence
No MD5 check before delete	<code>SyncService.ts</code> lines 359–375	2026-08-07	A damaged download that returns HTTP 200 is stored as the capture and the device's good copy is deleted. The loss would not be noticed

Gap	Where	Since	Consequence
Shared default token	AppConfig.ts line 52; config.py line 22	2026-07-22 (device), 2026-07-23 (app)	Any client on the network that sends the default token can list, download and delete captures on a unit that has not been given its own token
Plain HTTP	app.json line 24; config.py line 14; app.py line 33	2026-08-10 (app setting)	Another device on the same network can read the audio, transcripts and token, and could alter a recording

5.2 The released app

Both released builds of the app (v1.1.0) carry all three gaps. Their manifests set `usesCleartextTraffic` to `true`, their bundles contain the default token, and they contain the download code written on 2026-08-07 but not the MD5 comparison it replaced.

Table 5.2: Released app v1.1.0, both builds (measured, `data/raw/release_apk_inspection_2026-10-22.csv`)

Check	arm64-v8a	armeabi-v7a	Meaning
Manifest: <code>usesCleartextTraffic</code>	true	true	Plain HTTP allowed (Gap 3)
Bundle contains <code>vaha-dev-2026</code>	yes	yes	Default token compiled in (Gap 2)
Bundle contains the download code of 2026-08-07	yes	yes	The sync code described in Table 3.1
Bundle contains the MD5 comparison's message	no	no	The comparison is not in the shipped app (Gap 1)

5.3 The device server, run on a test machine

The server behaved as its code reads. With the default settings it refused requests without the token and accepted the default token from any client, including a request that deleted a capture. With `VAHA_AUTH_ENABLED=false` it accepted the same requests with no token at all. It did not accept a TLS handshake in either configuration. The downloaded audio matched the MD5 in the capture list each time; on the test machine nothing sits between client and server to damage it.

Table 5.3: Device server access test, one run per case (measured, `data/raw/server_access_test_2026-10-22.csv`)

Settings	Request	Token sent	Result
Default	Health check	none	200
Default	List captures	none	401
Default	List captures	wrong token	401
Default	List captures	default token	200
Default	Download a capture	default token	200; MD5 matches the list
Default	Delete a capture	none	401; capture kept
Default	Delete a capture	default token	200; capture folder removed

Settings	Request	Token sent	Result
Default	TLS handshake on the server's port	—	refused
Check switched off	List captures	none	200
Check switched off	Download a capture	none	200; MD5 matches the list
Check switched off	Delete a capture	none	200; capture folder removed
Check switched off	TLS handshake on the server's port	—	refused

The server was bound to the test machine itself, so the test shows which requests it accepts, not who can reach it on a real network. On the device it listens on all network interfaces (`config.py` line 14).

SECTION 6

Analysis and discussion

6.1 Results against the requirements

Table 6.1: Requirement conformance

ID	Criterion	Result	Status
R1	Local-only sync	The app downloads from the device's local address; no cloud service is in the sync path	Met
R2	Verified before purge	No MD5 comparison in the code or the released app; the device copy is deleted after HTTP 200	Not met
R3	Access limited to the paired app	The device checks a token, but every unit and app share the same default, and the check can be switched off; both confirmed on the test machine (Table 5.3)	Partly met
R4	Protected in transit	Plain HTTP; the released app allows it and the server refuses TLS (Tables 5.2 and 5.3)	Not met

R1 holds for sync. The README's wider statement that "no audio or text ever touches the cloud" does not hold when the optional Groq fallback is on, since it sends audio to `api.groq.com` (`LocalWhisperTranscriptionService.ts` line 66), or when Notion export is on. The README should say that these features are optional and send data off the device.

6.2 Why the missing MD5 check matters most

Because the device deletes its copy after sync, a damaged transfer is not a recoverable error: the handset's copy becomes the only one. HTTP 200 shows that the server sent a response, not that the file on the handset matches the recording. Whether `downloadAsync` can return HTTP 200 with a short or altered file has not been tested; R2 exists so that the answer does not matter. Until the check is back and tested, the README's "zero corrupted transfers" cannot be supported by the code. The same statement appears in the project report of 2026-08-17 (`docs/VAHA Report.pdf` , reference 1), and the 120 captures in Figure 2 were all reported as synced.

The session in Figure 2 has no recorded date. Its screenshots were added to the repository on 2026-08-17, so it ran on or before that day, and the app's sync service has existed since 2026-07-22 (Appendix B). If it ran before 2026-08-07, the app still made its fail-open MD5 comparison; if after, it made none. The record does not say which, so the screenshot cannot show that the checksum claim was tested.

To give the server side of the path a dated record, 120 synthetic captures were run through the unchanged device server on loopback with the default token (section 4.2, step 8). All 120 downloads returned HTTP 200, all 120 bodies matched the server's `audio_md5`, and all 120 delete requests returned HTTP 200, removing the device copies (measured, `data/raw/dated_120_capture_session_2026-10-22.csv`). This shows that the server serves intact files and deletes on request; it does not run the app, so it says nothing about what the app checks. That the app makes no comparison is a property of its code (Gap 1), not a result of this run. On a handset, each of these 120 would be marked synced on HTTP 200 alone.

Gap 1 is also a lesson in how integrity checks disappear. Nobody removed the check on purpose: it lived inside the download code, and when the download was rewritten for an unrelated reason, the check went with it. Nothing failed, because a missing check and a passing check look the same from outside.

6.3 MD5 protects against accidents, not attacks

An MD5 comparison catches accidental damage in transfer. It does not stop deliberate tampering on this link: the expected MD5 arrives in the capture list over the same unencrypted connection as the audio, so anyone who can alter one can alter both. For accidental damage MD5 is adequate; SHA-256 costs little more and avoids the question. Protection against tampering has to come from the transport (R4). Closing Gap 1 does not close Gap 3, and closing Gap 3 does not replace the MD5 check.

6.4 Closing the gaps

The code below is a sketch of each fix, written against the current code. It has not been built or tested yet.

Gap 1: verify, record, then delete. The order of the last three steps is what keeps a recording safe. The app checks the file, writes its own record, and only then asks the device to delete its copy. Any failure in the check fails the transfer, which the existing retry loop then downloads again:

```
// After the download has returned HTTP 200
const info = await FileSystem.getInfoAsync(localUri, { md5: true });
const verified = info.exists && !!info.md5 &&
  info.md5.toLowerCase() === metadata.audio_md5.toLowerCase();
if (!verified) {
  await FileSystem.deleteAsync(localUri, { idempotent: true });
  return Result.fail(HttpDeviceTransport.createChecksumError(metadata.transaction_id));
}
// Record the capture first, then ask the device to delete its copy
await captureRepo.update(captureId, { syncState: "synced" });
await transport.delete(`/captures/${metadata.transaction_id}`);
```

Two further measures make the order hold over time. A test that serves a file whose MD5 differs from the listed value, and expects the transfer to fail and the device copy to remain, would have caught Gap 1 when the download was rewritten. And if the device has room, it can keep a synced capture for a few days before deleting it, so that a missed fault is recoverable.

Gaps 2 and 3: a per-device secret and TLS without a certificate authority. A device on a home network has no domain name, so it cannot get a certificate from a public authority. The pairing step can carry the trust instead. On first boot the device generates a key pair and a self-signed certificate. At Bluetooth LE pairing, which needs the phone beside the device, the device gives the app its certificate's SHA-256 fingerprint and a per-device token; the design already plans to derive that token from a secret exchanged at pairing. The app keeps both in secure storage, sends the per-device token instead of the default, and accepts the device's HTTPS certificate only if its fingerprint matches the stored one. With TLS in place, `usesCleartextTraffic` can be switched off. On the device the change is small:

```
import hmac

def token_ok(presented: str, paired: str) -> bool:
    # Constant-time comparison; release builds have no switch that turns this off
    return hmac.compare_digest(presented.encode(), paired.encode())

uvicorn.run(app, host=HTTP_HOST, port=HTTP_PORT,
            ssl_certfile=DEVICE_CERT, ssl_keyfile=DEVICE_KEY)
```

The app side is harder. React Native's built-in HTTP stack trusts only the certificate authorities installed on the phone, and each device's certificate is known only at pairing, so build-time configuration is not enough. The fingerprint has to be checked in code, through a native module. Encryption on the home Wi-Fi does not remove the need for any of this: it keeps outsiders off the network, but every device that has joined the network is inside it.

6.5 Comparison with ETSI EN 303 645

ETSI EN 303 645 (reference 2), the baseline for consumer IoT security, is used here as a reference, not as a conformity assessment.

Table 6.2: Sync path against ETSI EN 303 645 provisions

Provision	What it asks	Sync path today
5.1 No universal default passwords	Credentials unique per device or set by the user	One default token for all units (Gap 2)
5.5 Communicate securely	Best-practice cryptography for communication	Plain HTTP (Gap 3)
5.8 Ensure that personal data is secure	Personal data protected in transit between device and associated services, including the companion app	Voice recordings and transcripts cross the network unencrypted (Gap 3)

SECTION 7

Limitations and threats to validity

- **No handset test yet.** The app's sync has not been run on a handset under test. The consequences in Table 5.1 for Gap 1 are what the code allows, not events that have been observed.
- **Server tested off the device.** The server code ran unchanged, but on a development machine, bound to that machine, with synthetic captures and one run per case (Table 5.3). Its access decisions do not depend on the hardware; its timing and network behaviour on the UNO Q were not tested.
- **Release inspected by its contents.** Table 5.2 shows which code the released app contains, found by text in its bundle, not how that code behaves when run.
- **One code state.** The paper covers the code as of 2026-08-17; the scripts and data added on 2026-10-22 did not change it. Units built from other states may behave differently. According to the development team (R. Bonthalakoti, 2026-10-22), no unit has shipped with its own token or with the check switched off (`VAHA_AUTH_ENABLED=false`); every unit in use therefore has the shared default. This is not recorded in the repository.
- **Sketches untested.** The code in section 6.4 has not been built or run.
- **Performance measured on loopback only.** The README's sync figures (Appendix C) have no recorded method. The timing in Table C.2 is from one run of 10 downloads on a development computer's loopback, with no radio link, so it neither confirms nor contradicts them.
- **Requirements set after the code.** R1 to R4 come from the README and were written down for this paper.
- **Severity is engineering judgement.** There is no formal threat model or vulnerability scoring.

SECTION 8

TRL assessment

TECHNOLOGY READINESS: WHERE THIS WORK SITS

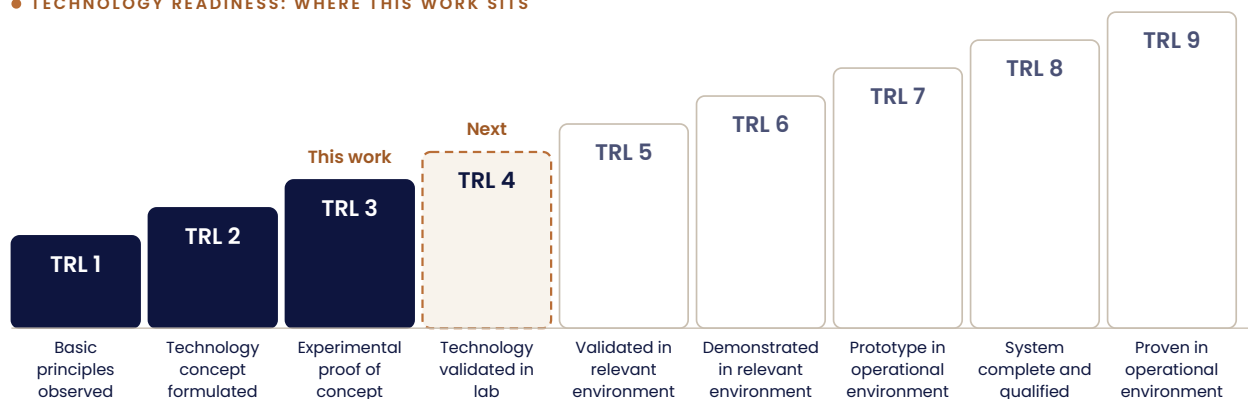


Figure 6 Where this work sits on the EU Horizon TRL scale. Definitions from the Dharanova publishing guidelines (EU Horizon TRL scale, ISO 16290 criteria, reference 3).

TRL demonstrated	Below TRL 4. The evidence supports an experimental proof of concept (EU Horizon TRL 3) for the sync path
Integration level	Subsystem: device server, transport, and app sync and storage together
Fidelity	Prototype: real device, real app, shipped code
Environment	Lab, local network
Evidence	Section 5: the code, the released app, and the device server run on a test machine
Why not TRL 4	TRL 4 requires validation in the lab <i>against defined criteria</i> . No criteria were set before the work, the sync has not been tested end to end on a handset, and R2 and R4 are not met
Gaps to TRL 4	Set limits first, then close each requirement and test it, as shown in Figure 7
Gaps beyond TRL 4	TRL 5 needs the relevant environment defined (a candidate: the home Wi-Fi networks VAHA will ship on, with a second device on the network trying to read or alter traffic) and the sync path tested there

● PATH FROM THIS WORK TO TRL 4

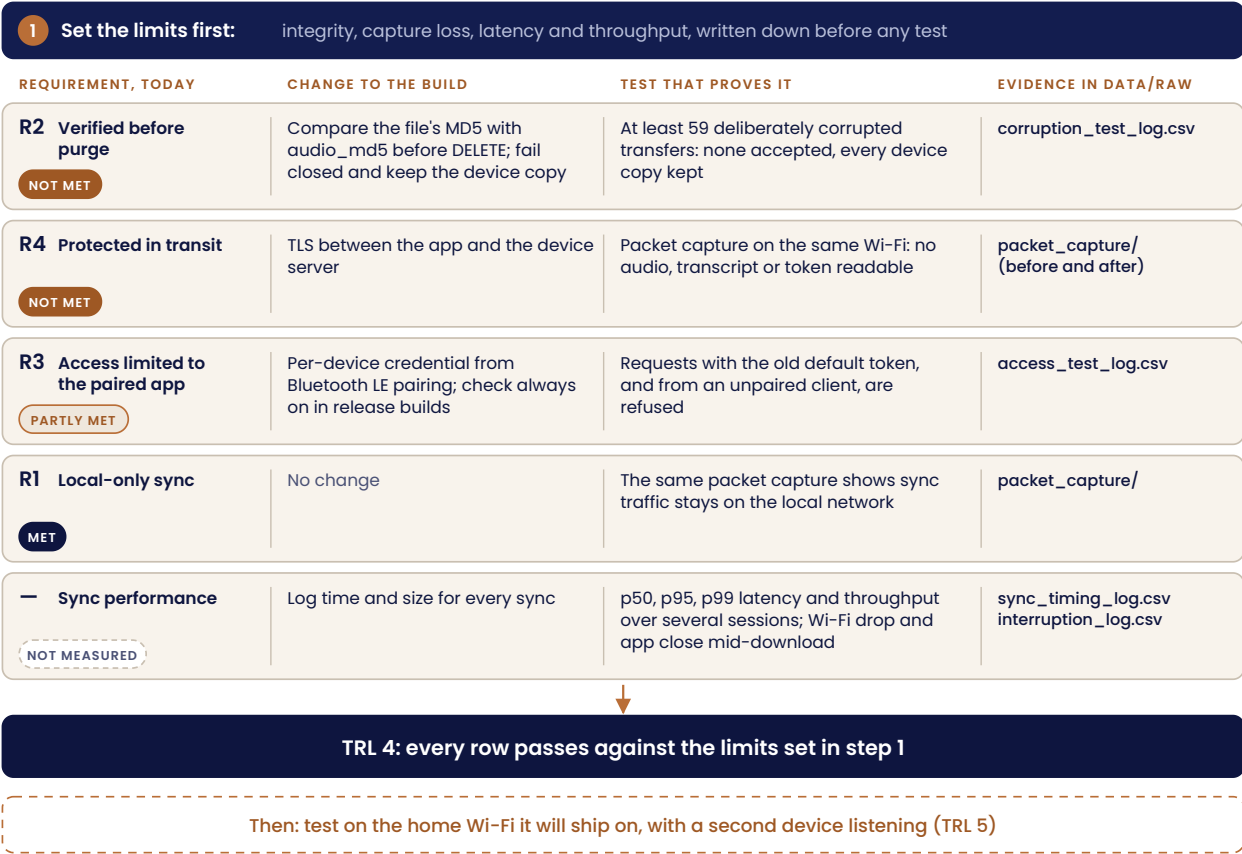


Figure 7 Path from this work to TRL 4, one row per requirement in priority order: its status today (Table 6.1), the change to the build, the test that shows it is met, and the data the test leaves in `data/raw`.

 SECTION 9

Conclusions and next steps

The sync path has the right shape for a privacy-first device: the app pulls captures directly from the device over the local network, with no cloud service in between. The controls that should make it safe are not yet in place, and only local-only sync of the four requirements is met. The missing MD5 check is the most urgent, because it can lose recordings for good and the README claims the opposite. The sync path is a proof of concept, below TRL 4.

Next steps, in the order of Figure 7:

1. **Set measurable limits** for integrity, capture loss, sync latency and throughput.
2. **Restore the MD5 check** as in section 6.4, and correct the README's checksum claim until it is tested.
3. **Corrupt at least 59 transfers on purpose** (calculated: zero failures in 59 trials shows at least 95 % rejection at 95 % confidence); none may be accepted and every device copy must be kept.
4. **Secure the link** with the per-device token and pinned TLS from section 6.4. Confirm that the old token is refused, and with a packet capture that nothing can be read and that sync traffic stays on the local network.
5. **Measure the sync:** p50, p95 and p99 latency and throughput with n over several sessions, plus transfers interrupted by a Wi-Fi drop or the app closing.
6. **Publish the results as a new version** of this paper.

KEY POINT

Checklist for local-only sync

- **Verify, record, then delete.** The sender deletes its copy only after the receiver has checked the hash and written its own record.
- **Fail closed.** If the check cannot run, the transfer has failed.
- **Test the check.** A test with a deliberately wrong hash stops a refactor from removing it silently.
- **Know what the hash is for.** MD5 or SHA-256 catches accidental damage; only an encrypted, authenticated link stops tampering.
- **No shared default credentials.** Issue a per-device secret at pairing, compare it in constant time, and leave no switch that turns authentication off in release builds.
- **Encrypt the local link.** Use TLS with the device's certificate pinned at pairing; Wi-Fi encryption does not protect against other devices on the same network.
- **Treat recordings and transcripts as personal data** on the network, not only in storage.
- **Break it on purpose.** Corrupt transfers, drop Wi-Fi, close the app mid-download and replay old tokens.
- **Report distributions, not single averages.** Give p50, p95 and p99 with n for latency and throughput.

● SECTION 10

References

1. Dharanova, *VAHA* source repository, <https://github.com/dharanova-code/vaha>. README sections "Verification & Real Bathroom Noise Results" and "Performance Metrics"; source files as listed in Appendix A.
2. ETSI EN 303 645 V2.1.1 (2020-06), *Cyber Security for Consumer Internet of Things: Baseline Requirements*.
3. ISO 16290:2013, *Space systems: Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment*.
4. Scripts and data for this paper, in the VAHA source repository (reference 1): `analysis/` and `data/`, added 2026-10-22.

APPENDICES

Appendices

A. Source locations

All paths are in the VAHA source repository (reference 1). Line numbers refer to the repository as of 2026-08-17.

Table A.1: Files and lines cited

File	Lines	What is there
mobile/src/features/sync/services/SyncService.ts	59, 79–120	Capture list request; local rows and queue jobs
	190, 202	Capture marked synced
	276–313	Retry loop
	336–375	Download, HTTP 200 test and delete request
	399–403	Retry wait calculation
mobile/src/features/devices/constants/ApiCompatibility.ts	43–49, 58	Retry constants; token used by the app
mobile/src/core/config/AppConfig.ts	52	Default device token
mobile/src/features/devices/transport/HttpDeviceTransport.ts	458–465	verifyChecksum() placeholder
mobile/src/features/captures/services/LocalWhisperTranscriptionService.ts	66	Optional Groq transcription request
mobile/app.json	24	usesCleartextTraffic: true
python/server/config.py	14, 19–25	Listen address; default token; switch for the token check
python/server/middleware/auth.py	33, 52	Token check and its bypass
python/server/app.py	33	Server start, no TLS
python/server/routes/captures.py	80, 85, 93	List, download and delete routes
python/server/services/capture_service.py	142, 166–169	MD5 computed and written to checksum.md5
release/vaha-companion-v1.1.0-*.apk	—	Released app builds inspected in section 5.2

B. When each behaviour was introduced

Table B.1: Sync path history, from the repository

Date	Change
2026-07-22	Sync engine, device server and the app's sync service created together. The device's default token and token check are added, and the sync service compares each download's MD5 with <code>audio_md5</code> , failing open
2026-07-23	Same default token added to the app's configuration
2026-07-27	App folder moved from <code>app/mobile</code> to <code>mobile</code> ; no change to sync behaviour
2026-08-07	Download moved to <code>downloadAsync</code> to fix failures on large WAV files; the MD5 comparison and its error path removed
2026-08-10	Plain HTTP enabled in the Android app "for local HTTP hardware communication"
2026-08-17	Last change to the code; the three gaps are still present
2026-10-22	Scripts and data for this paper added in <code>analysis/</code> and <code>data/</code> ; no change to the code

C. Performance figures quoted in the product README

These figures are reported in the README. No script, log or note found in the repository states a method, sample size or spread for any of the four.

Table C.1: Reported sync performance (README, method unknown)

Metric	Value	Source
Sync-initiation latency	about 250 ms average per capture	README (reference 1)
Transfer throughput, local Wi-Fi	about 1.5 MB/s	README (reference 1)
SQLite write latency, app	about 5 ms per transaction	README (reference 1)
WebSocket telemetry latency	under 10 ms	README (reference 1)

For comparison, the download was timed on the unchanged server code on a development computer's loopback (section 4.2, step 8): one run of 10 downloads of an 8 s synthetic capture, 768,044 bytes each. Loopback has no radio, distance or contention, so these figures say nothing about a handset-to-device transfer over Wi-Fi; they show that the measurement can be scripted and repeated. With $n = 10$, p95 and p99 are not given.

Table C.2: Loopback download timing, development computer (measured, $n = 10$, one run)

Metric	Mean	SD	Min	Median	Max
Time to first byte, ms	22.1	13.5	3.1	24.2	50.2
Throughput, MB/s	39.5	28.9	14.4	27.4	111.3

Source: `data/raw/sync_performance_test_2026-10-22.csv`, summarised by `analysis/summarise_sync_runs.py` into `data/processed/sync_performance_summary.csv`. Throughput is bytes divided by total download time, in millions of bytes per second. The first download was the

slowest; the cause of the spread was not investigated. Step 5 of section 9 runs the same measurement on the UNO Q over Wi-Fi, across several sessions.

D. Revision history

Version	Date	Author	Changes
v0.1	2026-10-19	Ritesh Bonthalakoti	First draft
v0.2	2026-10-22	Ritesh Bonthalakoti	Restructured to the publication template
v0.3	2026-10-22	Ritesh Bonthalakoti	Integrity gap corrected after tracing the full sync path: the app makes no MD5 comparison and deletes the device copy after HTTP 200. Device server covered. Gaps restated as known gaps of the current build and moved to Results. Section 6.4 on closing each gap and a checklist for local-only sync added. Requirement R4 and ETSI EN 303 645 comparison added. README performance figures moved to Appendix C. Released app inspected and device server run on a test machine (sections 5.2 and 5.3). Sync-path, TRL and path-to-TRL-4 diagrams added
v1.0	2026-10-22	Ritesh Bonthalakoti	First published version. 120-capture server run and loopback download timing added (section 6.2, Table C.2). Handset details and the Figure 2 session date stated as not recorded. Token use in shipped units confirmed with the development team. Appendix B corrected: the app's fail-open MD5 comparison dates from 2026-07-22, not 2026-07-27. Scripts and data published in the VAHA repository (reference 4). Internal R&D log removed as a source; the README figures it repeated are cited from the README alone



We build intelligence into everyday objects.

● HOW TO CITE

Ritesh Bonthalakoti. *Device-to-app sync for VAHA: architecture, and the integrity and security of the current build*. Dharanova whitepaper, v1.0, 2026.

dharanova.com · info@dharanova.com