2024-2025

INFO-Y115 - Secure Software Design & Web Security

Computer Project: Report

BOTTON David 615056

Pr. Absil R.

August 2025

Contents

ABSTRACT

This report details a security-focused infrastructure along its implementation in an end-to-end encrypted, PKI-based dashcam system. To be presented in this document; a quick-start guide with screenshots showing how users navigate the application, a systematic review of all components with an emphasis on the security aspect, the limitations and impediments encountered, a Q/A list resuming the system's security posture. Public `git` *repository and initial setup available at the GitLab README.*

1 Security Overview

1.1 Architecture

SecCam runs each service in a dedicated Docker container and anchors trust in a local PKI with step-ca. The PKI bootstrap script generates a root and intermediate CA, then issues 2048-bit RSA leaf certificates for the Nginx gateway, the application server and other internal helpers; it also prepares an optional PKCS#12 bundle for full mutual-TLS. Inside the cluster, every socket is negotiated with service certificates that chain to this intermediate CA, so impersonation requires the CA's signing key, not just an access to a single host key. For external clients the trust anchor is pinned: the React bundle embeds the CA-root SHA-256 fingerprint at build time, rejecting any server that does not present the expected public key.

1.2 Traffic

All traffic enters through an Nginx reverse proxy that terminates TLS 1.3 exclusively and disables session tickets to foil passive resumption correlation. HSTS, CSP, Referrer-Policy and X-Frame-Options headers are set globally. Upstream calls from the proxy to the application server reuse TLS with client authentication: Nginx presents its own certificate and validates the server's certificate and SAN before forwarding, eliminating blind-trust hop vulnerabilities, container-level impersonation, or transparent MITM. Also, rate-limiting and header sanitation add basic DoS and injection resistance. After TLS is established, the session identifier is carried in a signed JWT that the server sets as an encrypted, httpOnly, sameSite cookie, so its exfiltration is greatly mitigated.

1.3 Data

The browser creates its own cryptographic material before any upload: a RSA key pair for wrapping, a fresh AES-GCM content key, and a HMAC-SHA-256 key for explicit integrity separation. User profile fields and each video segment are encrypted with AES-GCM and tagged with the HMAC, only ciphertext and MAC leave the client. Both symmetric keys are wrapped with the user's RSA public key (RSA-OAEP) and stored alongside the ciphertext. The private key is saved only as an encrypted JWK inside the user-controlled .sec.json bundle and is decrypted into memory on demand; it never appears in cleartext on disk or in server logs. Because the content key is user-specific and the server never sees it unwrapped, a breach of the database yields nothing usable without the client's private key.

1.4 Streams

Trusted users undergo a dual-CSR workflow: one certificate asserts their organizational identity, the other authorizes platform access. After the server enacts the signature of both CSRs through the intermediate CA, these certificates allow normal users to share a stream key simply by wrapping it with the trusted user's certificate public key. Server stores only the wrapped blob and when the trusted party requests the video, the server hands over that blob and the encrypted chunks. Decryption happens entirely in the client, leaving the server permanently blind. Revocation reduces to deleting the stored wrapped key, after which the former recipient can no longer unwrap future AES keys. All sharing, verification and revocation therefore occur at the cryptographic edge.

2 Quickstart Guide

3 Pipelines

3.1 Registration

A Annex:

```
sd
```

Listing 1: caption