AY 2025/26

BLOCKCHAIN-BASED VIDEO SURVEILLANCE SYSTEM

IoT Security & Data Security Project

Samuele Sparno Lorenzo Di Palo

AGENDA

1. CONTEXT & CASE STUDY
2. REQUIREMENTS
3. SYSTEM ARCHITECTURE
4. DIAGRAMS
5. VULNERABILITIES
6. ERRORS & ANOMALIES

CONTEXT

A security camera records events when motion is detected.

These events generate photos and metadata that may be needed later, even long after.

It is important to reconstruct the event history and verify data integrity, without exposing sensitive content.

THE PROBLEM

In a forensic context, I need to prove that a photo has remained unchanged since it was taken.
With traditional storage, a file can be copied or modified, and proving it later is difficult.

THE SOLUTION

The photo is encrypted and stored locally on IPFS.
On-chain, I record only its hash and the camera’s signature.
When needed, I recompute the hash from the data and compare it with the on-chain value: if they match, integrity is verified.

CASE STUDY

Scenario: building entrance with a motion sensor that activates the camera when movement is detected.

Evidence produced: encrypted photo on IPFS (CID) and a signed SHA-256 hash recorded on-chain for future forensic verification.

SYSTEM ARCHITECTURE

PIPELINE END-TO-END
(EDGE → IPFS → BLOCKCHAIN)

EDGE LAYER	APPLICATION LAYER	STORAGE LAYER
ARDUINO + PIR motion trigger	**MQTT** Pub/Sub Messaging	**IPFS (local)** Ciphertext CID
ESP32-CAM Capture + Aes encryption	**GO RECEIVER** Hashing + Signing	**FireFly + Smart Contract** verify + record on-chain

FUNCTIONAL REQUIREMENTS

1

MOTION TRIGGER

The system must detect motion and automatically trigger a capture.

2

ON-DEVICE CAPTURE & ENCRYPTION

The system must capture the photo and encrypt it locally (AES-128-CBC + IV) before sending it.

3

SECURE DELIVERY VIA MQTT

The system must send an MQTT JSON payload with the encrypted photo and metadata (IV, identifiers, location).

4

IPFS STORAGE

The system must upload the ciphertext to IPFS and obtain the CID as a persistent reference.

5

ON-CHAIN INTEGRITY PROOF

The system must record on-chain the signed SHA-256 hash and use an anti-replay nonce to validate integrity and origin.

NON-FUNCTIONAL REQUIREMENTS

1

SECURITY & PRIVACY

The system must encrypt photos and store on-chain only hashes and minimal metadata, avoiding any plaintext content.

2

INTEGRITY

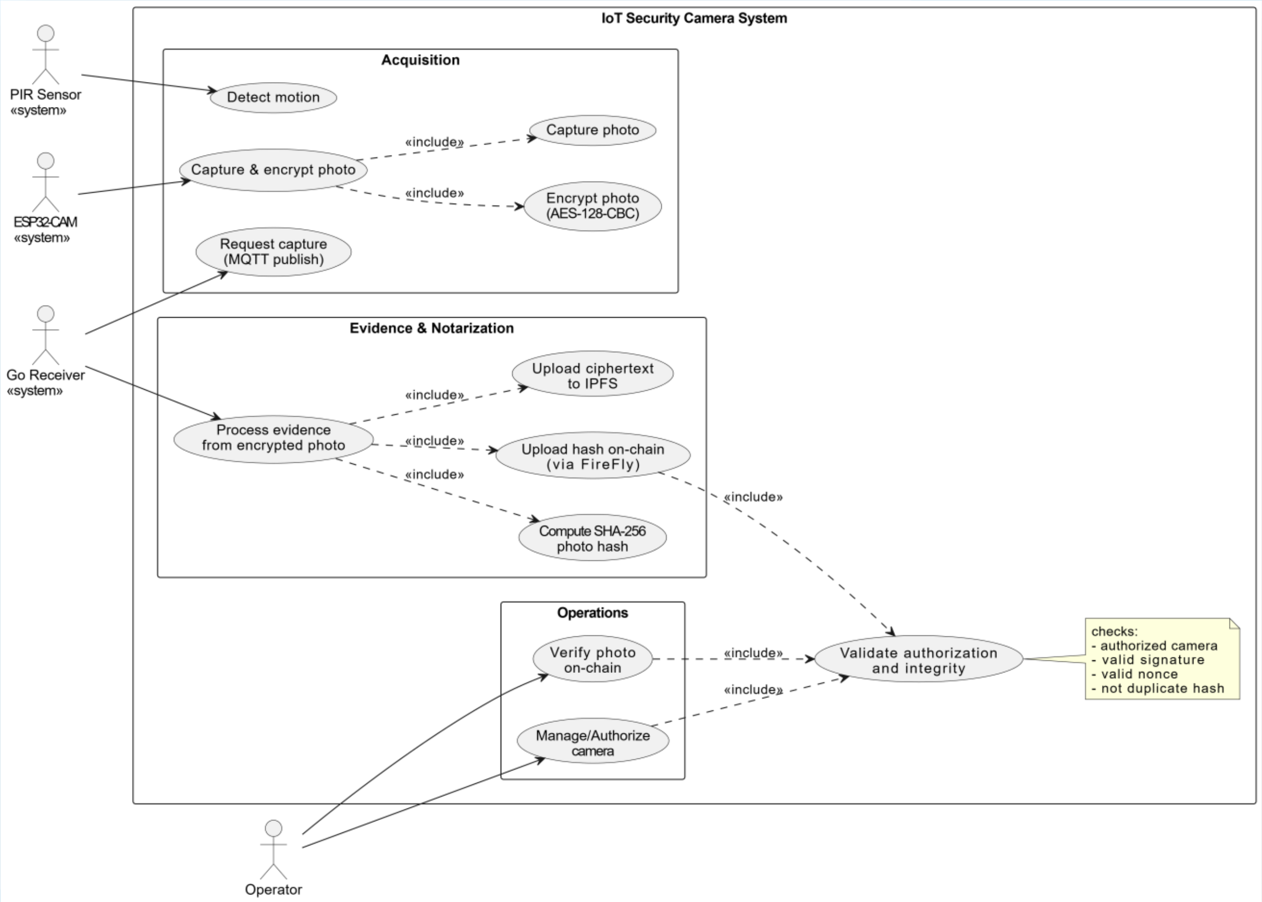
The system must support integrity verification by anchoring the hash on-chain as an immutable reference.

3

AUTHENTICITY & ANTI-REPLAY

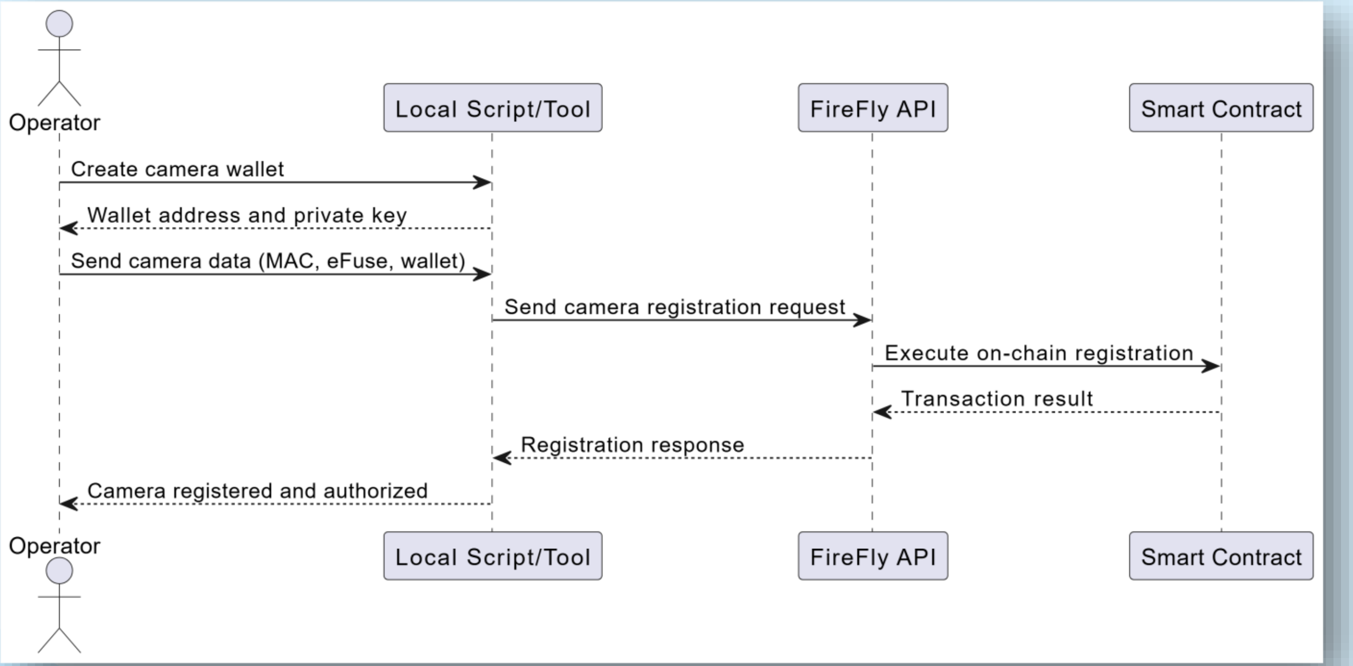
The system must accept events only from authorized cameras via digital signatures and block replay using a nonce.

USE CASE DIAGRAM



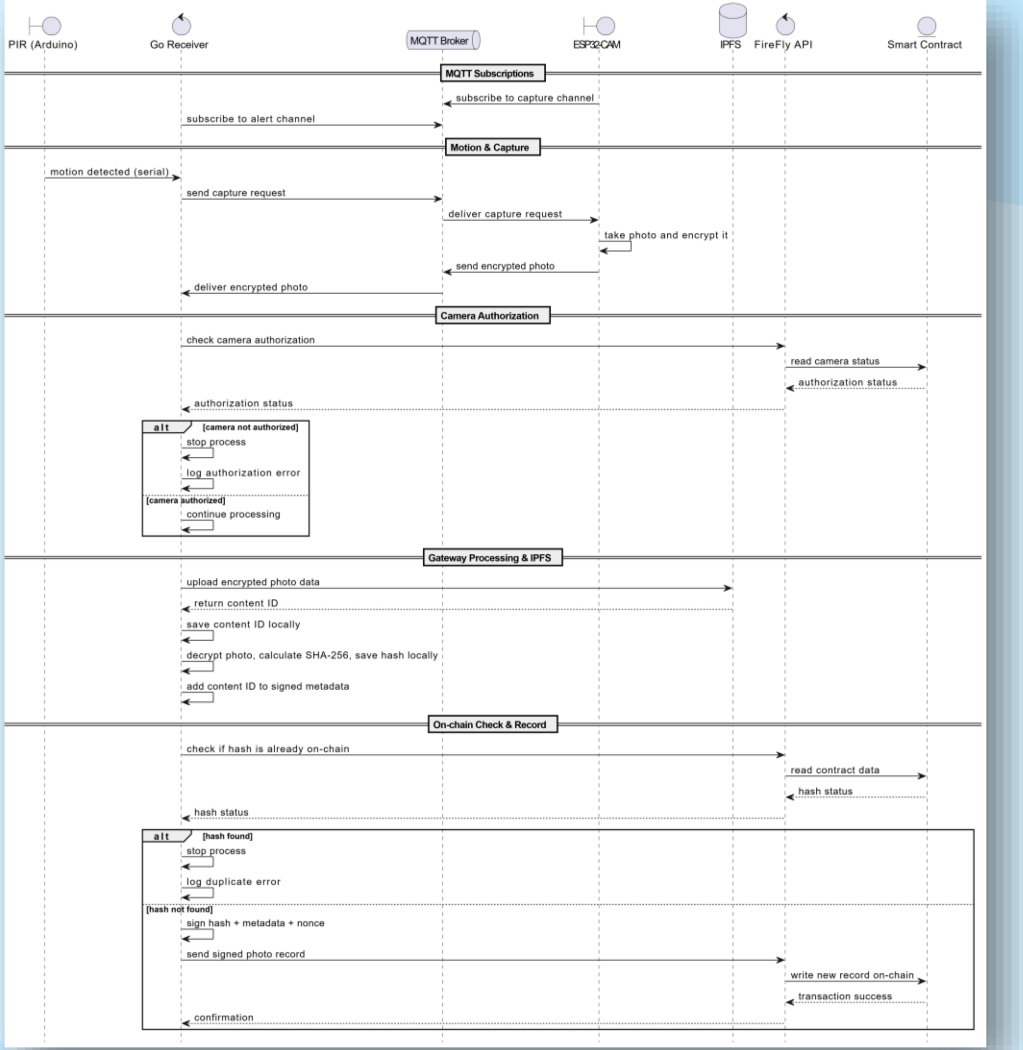
SEQUENCE DIAGRAM

Camera Registration


Operator
Local Script/Tool
FireFly API
Smart Contract
Create camera wallet
Wallet address and private key
Send camera data (MAC, eFuse, wallet)
Send camera registration request
Execute on-chain registration
Transaction result
Registration response
Camera registered and authorized
Operator
Local Script/Tool
FireFly API
Smart Contract

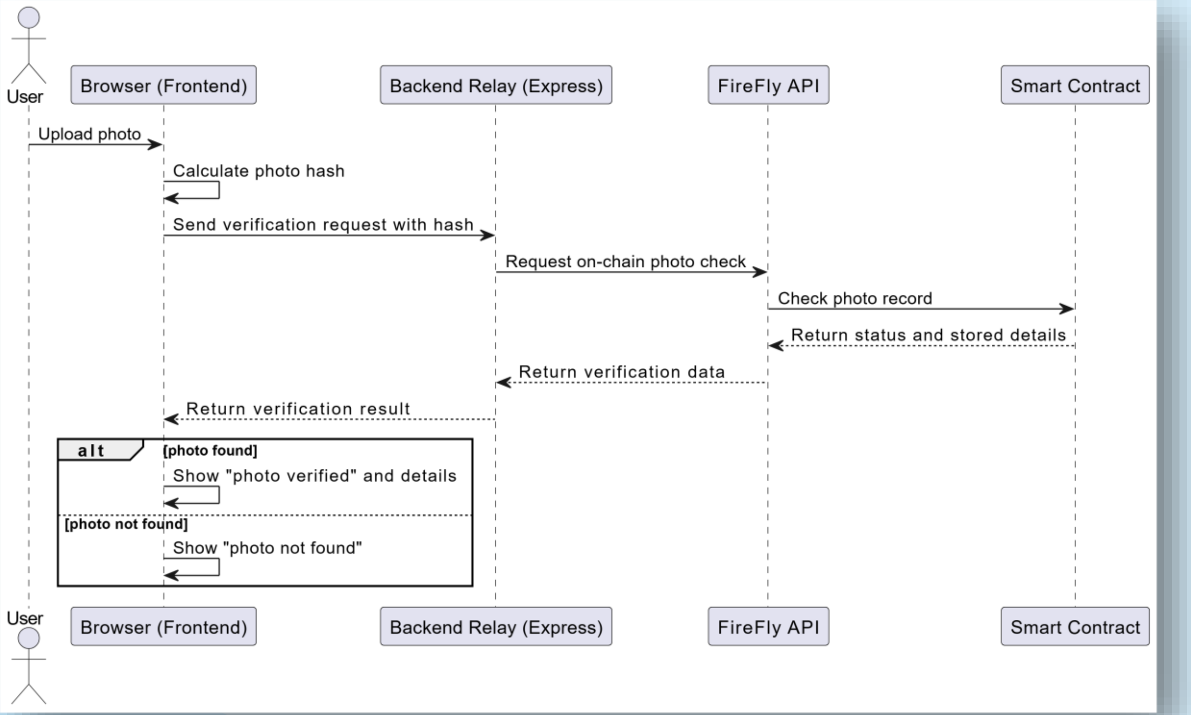
SEQUENCE DIAGRAM

Capture, Encryption, IPFS, and On-Chain Recording

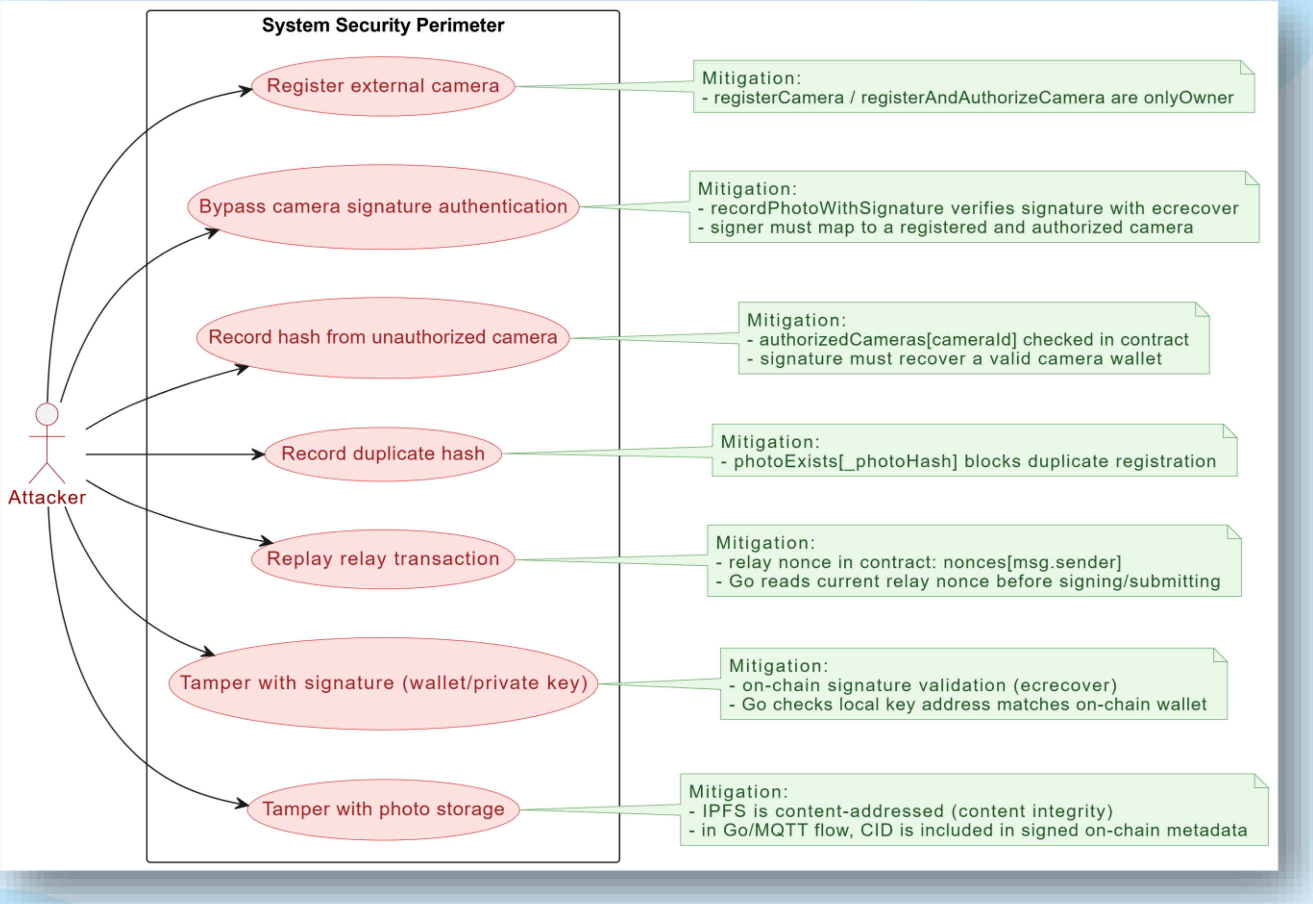


SEQUENCE DIAGRAM

Forensic Proof Verification



MISUSE DIAGRAM


System Security Perimeter
Attacker
Register external camera
Bypass camera signature authentication
Record hash from unauthorized camera
Record duplicate hash
Replay relay transaction
Tamper with signature (wallet/private key)
Tamper with photo storage

Mitigation:
- registerCamera / registerAndAuthorizeCamera are onlyOwner

Mitigation:
- recordPhotoWithSignature verifies signature with ecrecover
- signer must map to a registered and authorized camera

Mitigation:
- authorizedCameras[cameraId] checked in contract
- signature must recover a valid camera wallet

Mitigation:
- photoExists[_photoHash] blocks duplicate registration

Mitigation:
- relay nonce in contract: nonces[msg.sender]
- Go reads current relay nonce before signing/submitting

Mitigation:
- on-chain signature validation (ecrecover)
- Go checks local key address matches on-chain wallet

Mitigation:
- IPFS is content-addressed (content integrity)
- in Go/MQTT flow, CID is included in signed on-chain metadata

VULNERABILITY

TRAFFIC INTERCEPTION (MQTT)

ATTACK
MITM / LAN SNIFFING

An attacker intercepts the MQTT payload and tries to read the photo or its metadata.

DEFENSE

The photo is encrypted end-to-end (AES on-device). Even if traffic is intercepted, only ciphertext is obtained.

VULNERABILITY

CAMERA SPOOFING

ATTACK

FAKE CAMERA SENDS PHOTO/HASH

An attacker publishes to MQTT pretending to be the camera and tries to register fake photos or hashes.

DEFENSE

The contract records only events signed by an authorized camera.
If the signature or sender is not valid, the event is rejected on-chain.

VULNERABILITY

TRANSACTION REPLAY

ATTACK
REPLAY

An attacker reuses a previously valid transaction or signature to duplicate records and create fake events.

DEFENSE

Each event includes a unique nonce in the signed message. If the nonce has already been used, the contract rejects the transaction.

VULNERABILITY

STORAGE TAMPERING (IPFS)

ATTACK

TAMPERING / FILE REPLACEMENT

An attacker tries to replace the stored photo or make it point to different content.

DEFENSE

IPFS is content-addressed: if the content changes, the CID changes too.
In addition, I compare the on-chain SHA-256 hash to verify integrity for forensic purposes.

ERRORS & ANOMALIES

AUTHORIZATION ANOMALIES

Unauthorized camera

Only registered cameras can record hashes: the request is rejected on-chain.

ERRORS & ANOMALIES

NO DATA DUPLICATION

Photo already recorded (duplicate hash)

If the hash already exists, the event is ignored and no duplicates are created.

ERRORS & ANOMALIES

REPLAY

Invalid nonce (replay)

If the nonce is already used or not valid, the transaction is rejected.

ERRORS & ANOMALIES

CRYPTOGRAPHIC ANOMALY

Invalid signature

If the signature is invalid, the contract rejects the record and the event is not stored.

THANK YOU FOR YOUR ATTENTION

Project Repository

LORENZO DI PALO

L.DIPALO@STUDENTI.UNISA.IT

NF25500048

SAMUELE SPARNO

S.SPARNO@STUDENTI.UNISA.IT

NF22500034