# ECE 4301 Midterm - Secure Video Streaming using Raspberry Pi 5

**Introduction**

This project implements a real-time encrypted video streaming system between two Raspberry Pi 5 devices using Rust as the programming language. The goal was to stream live video securely. Instead of using simple passwords or weak encryption, this system performs symmetric encryption, asymmetric key exchange, and periodic rekeying to emulate real-world secure media streaming. The Raspberry Pi 5's ARMv8 crypto extensions (AES and PMULL) were enabled when available, and the system was also tested without hardware acceleration to evaluate performance differences.

**System Architecture**

Two Raspberry Pi 5 units were connected as one Pi acts as the sender, while the other acts as the receiver.

Sender Path

- Webcam captures frames
- Gstreamer preprocesses video input
- Frames are encoded
- AES-GCM encryption applied (Rust)
- Encrypted packets are transmitted using TCP
- Rekeying is performed periodically and logged

Receiver Path

- TCP listener receives encrypted packets
- AES-GCM decrypts and authenticates each frame
- Verified frames injected into Gstreamer
- Video decoded and displayed in real-time

**Packet Protocol**

Each packet sent across the network follows a structured format:

| Field | Purpose |
|---|---|
| Start byte | Marks beginning of the packet |
| Packet type | Data, public key, encrypted session key, or rekey trigger |
| Length (u32) | Payload size |
| Packet data | Encrypted video or key material |
| Timestamp | 17-byte clock snapshot |
| End byte | Packet termination marker |

**Security Design**

Asymmetric Key Exchange: Two methods were implemented

| Method | Notes | Performance |
|---|---|---|
| RSA-3072 | Classical PKI baseline | Slower (~300-500ms) |
| ECDH (P-256) | Used by default | Near-instant (<1 ms) |

Both devices generate temporary key pairs. Keys are never stored on disk and are wiped when the program exits. Rekeying occurs periodically.

Symmetric Encryption

- AES-128-GCM
- Packet counters prevent replay attacks
- Any integrity failure triggers rejection (none observed)

This approach mirrors secure media transport used by modern messaging/video systems.

**Measurement Setup**

Hardware

- Two Raspberry Pi 5 boards
- USB camera
- Power meter

Software

- Rust 1.79, Tokio async runtime
- GStreamer for media pipeline

Metrics Logged

- Frame rate
- Latency per frame
- Encrypted throughput
- CPU usage
- Handshake timing

**Results**

| **Metric** | **Sender** | **Receiver** |
|---|---|---|
| Avg FPS | ~14.9 | ~15.0 |
| Data Rate | ~9.6 Mb/s | ~9.3 Mb/s |
| CPU Usage | ~4.3% | ~4.8% |

No frame drops occurred during steady-state operation, and AES-GCM authentication did not fail. Average latency was approximately 14 ms. Hardware crypto acceleration was detected, but it did not significantly alter system performance because video encoding dominates CPU load.

**N-Node Scalability Model**

To evaluate scaling beyond one receiver, a streaming to N receivers is considered.

| Resource | Scaling |
|---|---|
| AES encryption | O(1) - encrypted once |
| Network transmission | O(N) - one stream per receiver |
| ECDH rekeying | O(N x ~1 ms) |
| CPU impact | Minimal until the network bottleneck |

**Discussion**

Key Observations:

- Real-time encrypted video streaming is sustainable on Pi 5 hardware
- Hardware acceleration exists, but the video encode load dominates total CPU time
- ECDH outperforms RSA by three orders of magnitude in latency

Limitations:

- Timestamp resolution limited true latency observation
- Clock skew between Pis occasionally produced false spikes
- System currently supports only one active stream

**Conclusion**

This project demonstrates that secure, low-latency video streaming can be achieved on low-power devices. By combining GStreamer, Rust, AES-GCM, and ECDH key exchange, we achieved real-time encrypted playback at ~15 FPS with negligible CPU load and strong security guarantees. These results demonstrate that modern public-key cryptography, authenticated encryption, and real-time multimedia can be combined effectively on low-cost embedded systems without compromising performance or security.