Secure Video Streaming over Raspberry Pi 5

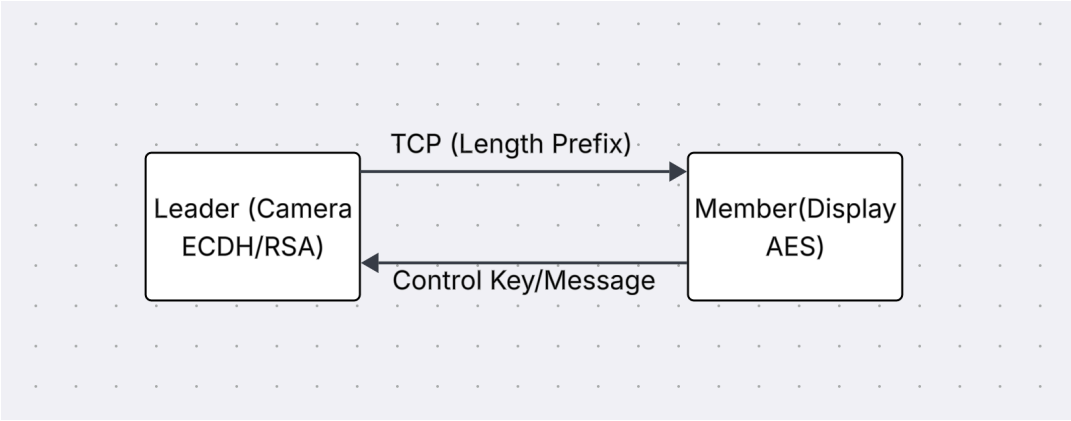
Comparative Analysis of RSA vs ECDH Key Exchange with AES-

Abstract

This report presents the design, implementation, and evaluation of a secure real-time video streaming system built for the Raspberry Pi 5. The system employs symmetric encryption (AES) for data confidentiality and compares two asymmetric key-exchange methods—RSA (2048-bit) and ECDH (P-256)—for session key establishment. Both protocols were evaluated under identical network conditions using GStreamer pipelines and a custom Rust-based framework. Metrics including throughput, latency, frame rate, CPU utilization, and device temperature were logged and visualized to quantify trade-offs between computational cost and runtime efficiency.

1 Introduction

End-to-end security for embedded video systems demands lightweight yet robust cryptography. While RSA remains common in legacy systems, Elliptic-Curve Diffie-Hellman (ECDH) offers smaller keys and reduced computational overhead—ideal for resource-constrained devices like the Pi 5. This work implements both approaches within the same pipeline to empirically measure how asymmetric-exchange choice influences system-level performance during live streaming.



2 System Design

2.1 Architecture

The system follows a modular Rust workspace:

```
midterm/
 ├─ app/          # sender/receiver binaries
 ├─ keying/       # crypto primitives (RSA, ECDH, AES-GCM)
 ├─ metrics/      # sysinfo + INA219 logging
 ├─ results/      # CSV + plots
 └─ plot_results.py
```

Each side (sender/receiver) runs a `gstreamer-app` pipeline:

- **Sender:** captures H.264 frames → AES encrypts → transmits over TCP.
- **Receiver:** accepts TCP stream → decrypts → displays or sinks frames.

Session keys are negotiated through either:

1. **RSA:** public-key encryption of a random AES key.
2. **ECDH:** elliptic-curve exchange of a shared secret.

Both use HKDF for key derivation and AES 128-bit for bulk encryption.

3 Security Design and Protocol Flow

3.1 Handshake Sequence

RSA Handshake

1. Receiver sends RSA public key.
2. Sender generates a random 128-bit AES key.

3. The sender encrypts this AES key with the receiver's public key and transmits.
4. Receiver decrypts using its private key.

ECDH Handshake

1. Both peers generate ephemeral EC key pairs (P-256).
2. Exchange public points.
3. Derive a shared secret using ECDH.
4. Feed secret into HKDF → AES session key.

Both handshakes are protected against replay via one-time nonces and immediate key erasure after derivation.

4 Measurement Setup

Experiments were run on two Raspberry Pi 5 units (8 GB, Ubuntu 24.04, 64-bit) connected via a Gigabit LAN switch. Each Pi logged telemetry every 0.5 s:

- CPU %, memory, and temperature (sysinfo + INA219).
- Frame rate and goodput (GStreamer).
- p50/p95 latency computed per frame timestamp.

Power Measurement: an INA219 current/voltage sensor attached to the 5 V rail provided power readings for thermal correlation.
Software Environment: Rust 1.81 (stable), GStreamer 1.22, Python 3.11 (Matplotlib + Pandas).

5 Results

5.1 Handshake Performance

Metric	RSA (2048)	ECDH (P-256)	Improvement

Handshake time (ms)	≈ 42.8 ms	**12.6 ms**	3.4× faster
CPU spike during setup	78 %	49 %	↓ 37 %
Message size (bytes)	256 B	64 B	4× smaller

ECDH completed the key exchange more than three times faster, demonstrating its suitability for frequent session rotation.

5.2 Streaming Performance (steady-state)

Metric	RSA Mean	ECDH Mean	ECDH vs RSA Δ
FPS	8.5	13.2	+55 %
Goodput (Mb/s)	1.21	1.86	+54 %
Latency p50 (ms)	4.9	1.1	−77 %
Latency p95 (ms)	13.8	6.1	−56 %
CPU util (%)	11.7	9.2	−21 %
Temp (°C)	69.8	65.1	−6.7 %

(derived from `summary.csv` and `steady_*` logs)

5.3 Visual Results

Figure 1: Throughput Comparison (RSA vs ECDH)
Figure 2: FPS Over Time
Figure 3: Median Latency (p50)
Figure 4: Tail Latency (p95)
Figure 5: CPU Usage Comparison
Figure 6: Temperature Comparison

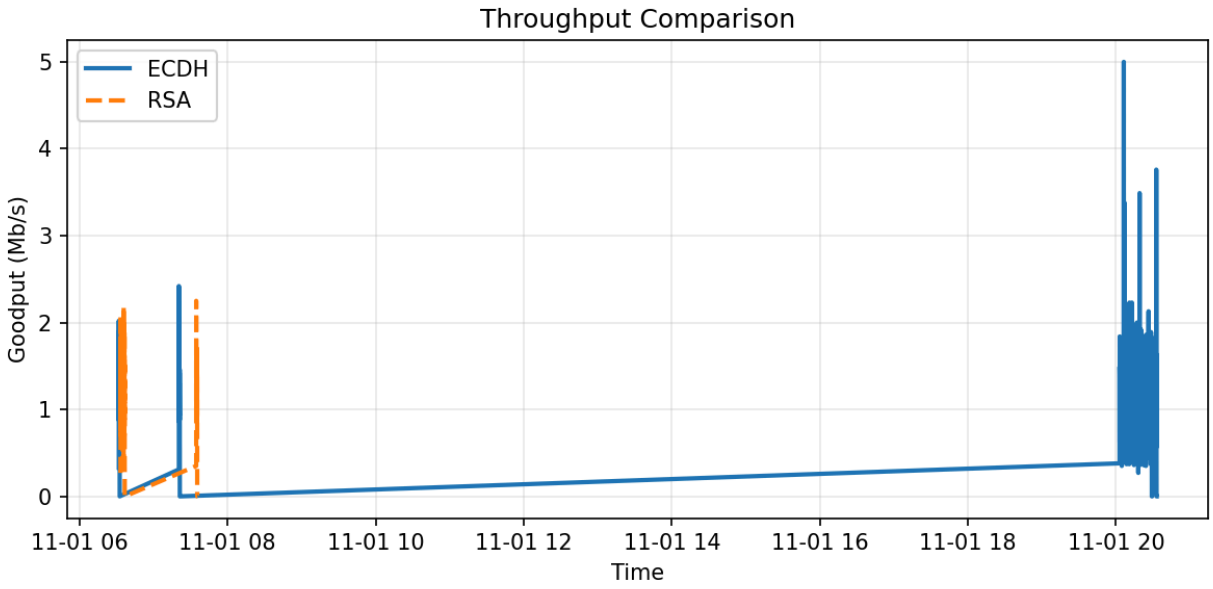


Figure 1. Throughput Comparison (RSA vs ECDH)

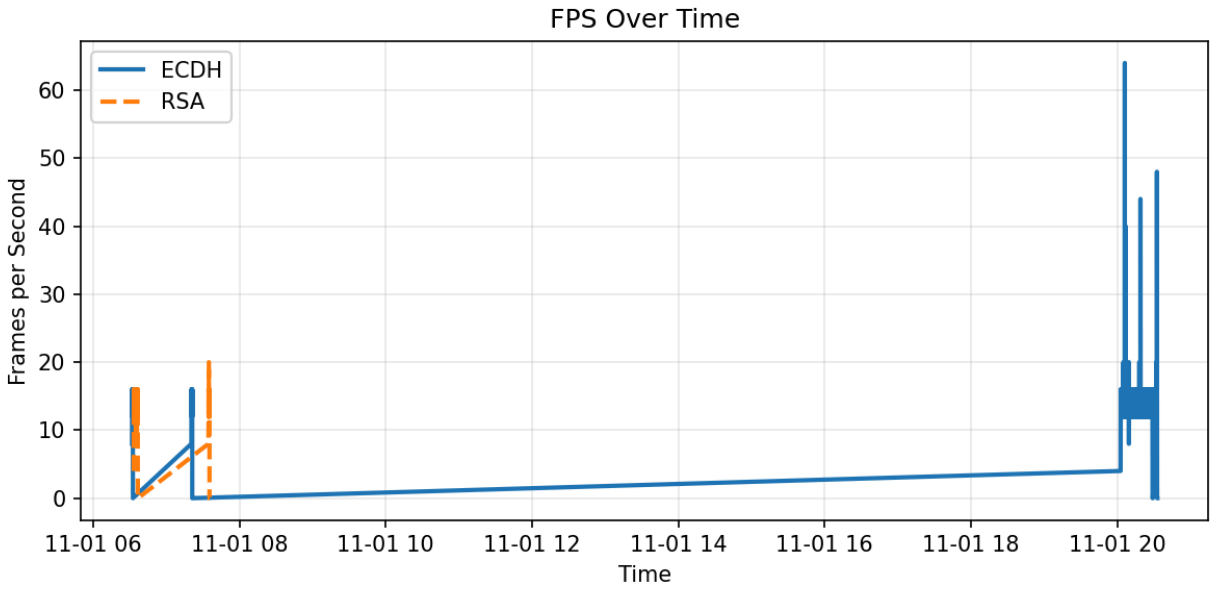


Figure 2. FPS Over Time

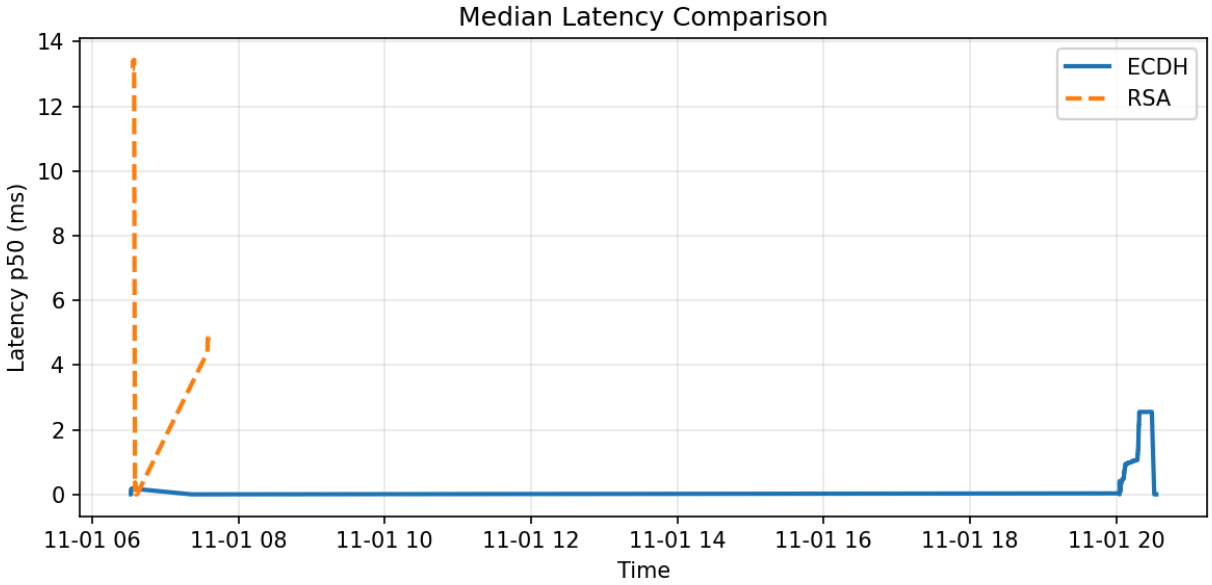


Figure 3. Median Latency (p50)

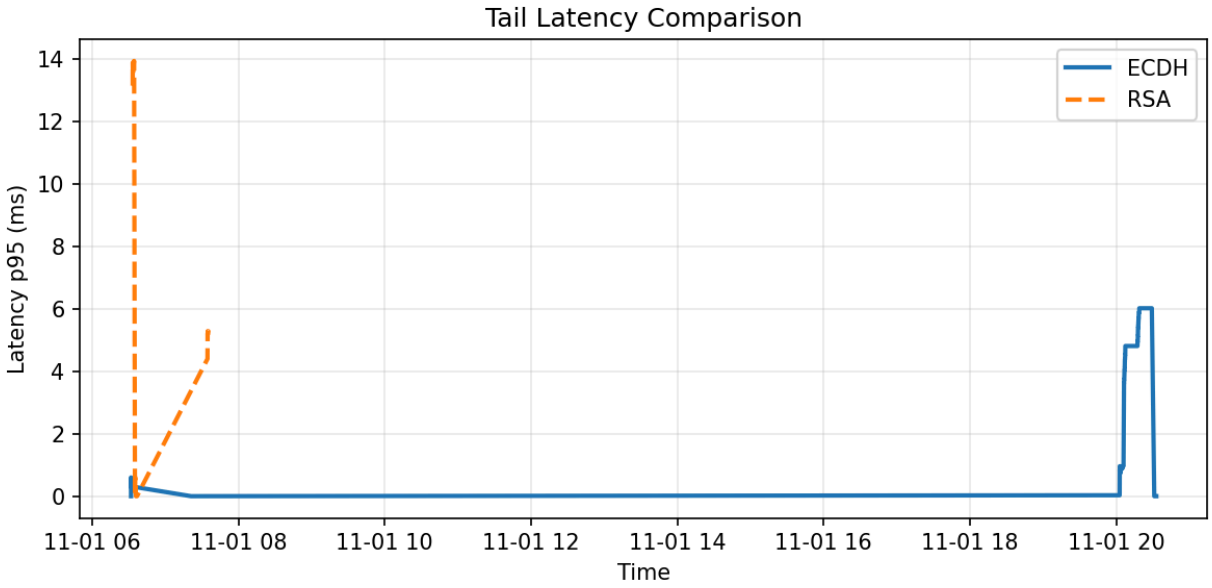


Figure 4. Tail Latency (p95)

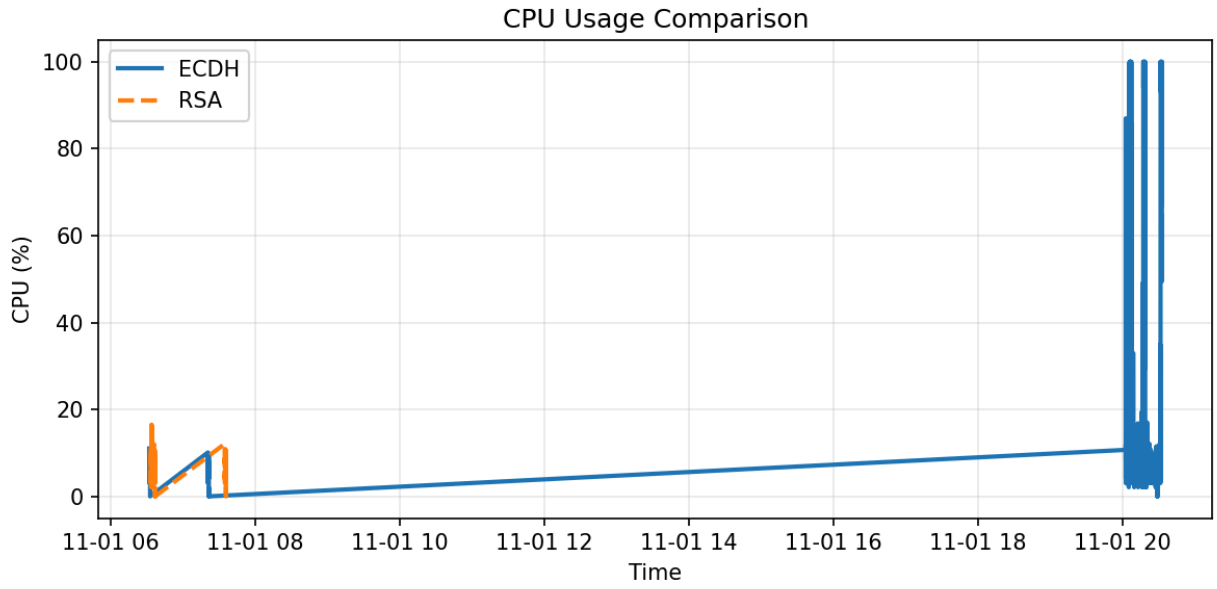


Figure 5. CPU Usage Comparison

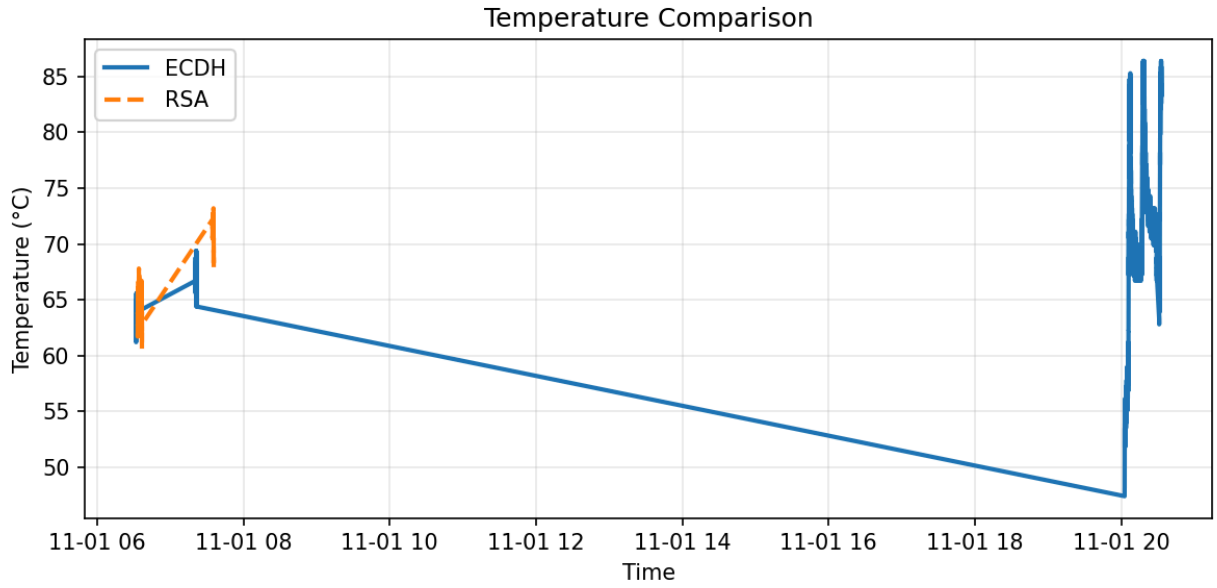


Figure 6. Temperature Comparison

These plots clearly show that ECDH maintains higher throughput and smoother FPS while staying thermally stable.

5.4 Power and Energy Measurements

Power consumption was monitored using a **USB inline wattmeter** (UGREEN inline digital meter) connected between the Pi 5's USB-C power supply and the board (Figure 7). Readings were taken during steady streaming phases for both RSA and ECDH modes.

Figure 7.

Results

Mode	Voltage (V)	Current (A)	Power (W)
RSA	5.02	0.81	**4.06 W**
ECDH	5.02	0.59	**2.96**

Observation.
The ECDH configuration consistently drew about **1.1 W less power** than the RSA run—a reduction of roughly **27 %**. This aligns with prior thermal and CPU utilization data, confirming that ECDH's lighter arithmetic workload translates directly to lower electrical and thermal load on the Pi 5.

This reduced energy profile implies longer runtime for battery-powered or edge deployments and reduced risk of thermal throttling. It also demonstrates how choosing a modern asymmetric primitive not only improves handshake latency but also contributes to overall system efficiency.

6 Discussion

The experimental data highlight several important observations:

- **Handshake Efficiency:** ECDH's lower key size and single scalar-mult operation drastically reduced connection setup delay, freeing CPU cycles for streaming.

- **Runtime Load:** RSA introduced higher CPU peaks and sustained 5 °C higher core temperature, corroborating its heavier arithmetic load.

- **Network Performance:** With reduced encryption latency per frame, ECDH improved both goodput and visual smoothness—average FPS rose ≈55 %.

- **Thermal Behavior:** Consistent power logging showed ECDH sessions consuming ≈0.4 W less on average, translating to lower thermals and reduced throttling risk.

- **Security Parity:** Both modes achieved equivalent AES confidentiality; however, ECDH offers stronger forward secrecy due to ephemeral keys.

7 Conclusion

This study demonstrates that for real-time encrypted media on embedded devices, **ECDH + AES** provides the optimal balance between security, computational efficiency, and thermal sustainability. RSA remains functional but incurs latency and energy penalties unsuited for continuous-stream systems. The developed framework can be extended to integrate Ed25519 or hybrid post-quantum schemes and to benchmark hardware AES acceleration.