Cal Poly Pomona

College of Engineering

ECE 4301.01

Fall 2025

Dr. Mohammed E. Aly

Midterm 1 - Secure Video Streaming with Raspberry Pi (Rust First)

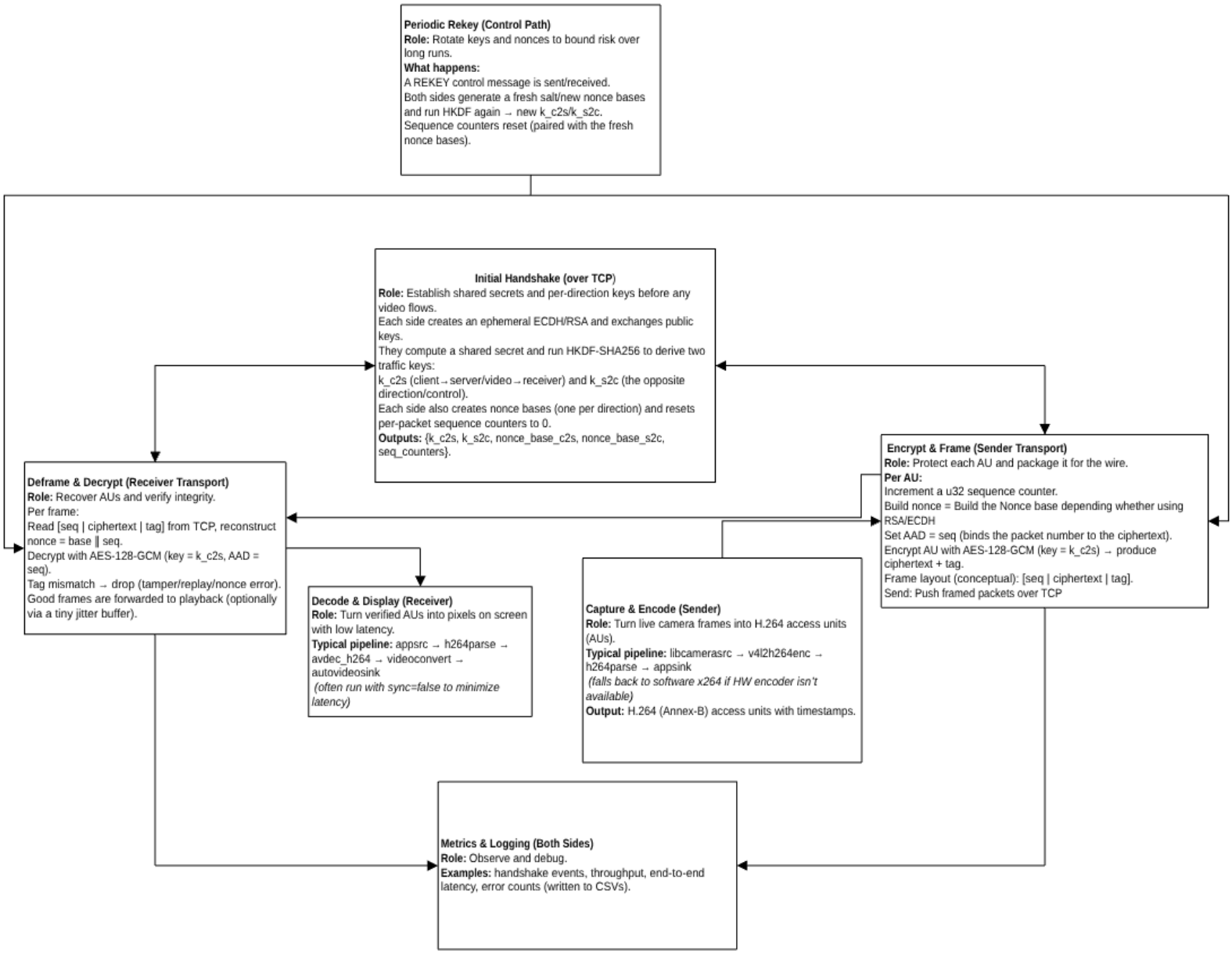
By Quinn Bell, Melvin Contreras, Aaron Tran

I. Overview

This report walks through all the necessary steps and gives context from setting up the experiments to the results given. Starting with the basic concepts of the program design, the differences between the ECDH and RSA variants, alongside the separate 3 node version. The benefits of AES-GCM and P256. The process of Streaming video data over ethernet connections. The experiments that were ran on 3 Raspberry Pi 5 models, each with some differences in peripherals, cooling, storage, and additional connected hardware.

II. Design

The secure video streaming project was developed to explore how modern cryptographic methods could be combined with efficient networking in Rust to build a confidential, authenticated, and low-latency communication channel. The system integrates cryptographic key exchanges using Elliptic Curve Diffie–Hellman (ECDH) and (Rivest-Shamir-Adleman) RSA. The design reflects practical engineering decisions intended to make encryption lightweight enough for real-time video while maintaining strong security boundaries. Each node operates as both a sender and receiver, establishing encrypted links to its peers, exchanging ephemeral public keys, and synchronizing periodic rekey events to prevent cryptographic reuse. The system is written in asynchronous Rust using the Tokio runtime, ensuring that concurrent encryption, network transmission, and logging tasks execute without blocking, thus maintaining throughput and frame synchronization even under continuous key rotation.



III. Security

Several deliberate security choices underpin this system's design. The selection of P-256 for ECDH provides a strong trade-off between computational cost and cryptographic security, allowing real-time encryption on embedded or low-power devices such as Raspberry Pi nodes. The use of HKDF ensures that all derived keys are domain-separated and context-bound,

preventing cross-protocol leakage. AES-GCM provides authenticated encryption with minimal performance overhead and is resistant to both bit-flipping and ciphertext malleability attacks. The system also incorporates strict sequence management to prevent nonce reuse and to guarantee that each ciphertext has a unique initialization vector. Furthermore, all rekey messages are authenticated and logged, ensuring that no unauthorized key rotations occur. The periodic rekeying not only extends forward secrecy but also minimizes exposure time if a key were ever compromised. Together, these mechanisms form a robust, layered defense that maintains confidentiality and integrity throughout continuous operation. Below is a diagram showcasing the protocol the program follows.

IV. Measurements & Setup

The measurement setup for this project was designed to assess both the cryptographic performance and system power consumption during active video transmission. The experiments were conducted using three Raspberry Pi nodes connected over a local network, each running the Rust-based streaming binary with logging enabled. A USB inline power meter was used on each node's power supply to capture voltage, current, and wattage data throughout streaming sessions. These readings were recorded during key exchange, steady-state transmission, and rekey events to identify the computational impact of cryptographic operations. CPU utilization and temperature were monitored in parallel using system metrics collected in CSV logs. Together, these measurements allowed evaluation of whether the chosen cryptographic primitives added meaningful overhead to the video pipeline. The results confirmed that encryption and rekeying introduced negligible delays relative to frame transmission intervals, validating the system's efficiency under real conditions. To be clear with the set parameters, below is the hardware specifications for each Pi used, though important to notate, the overall effect of the differences in hardware seemed to be negligible.

	Pi 1	Pi 2	Pi 3
Memory	16 Gb	16 Gb	8 Gb

Storage	128 Gb Micro SDXC Card	512 Gb NVME SSD Via NVME HAT	128 Gb Micro SDXC Card
Cooling	Ice Peak Cooler 3510	RPi Active Cooler SC1148	Argon Neo 5 Black
Camera	Arducam 5MP 1080p RPi 5 Camera	Arducam 5MP 1080p RPi 5 Camera	RPi Camera Module 3

V. Results

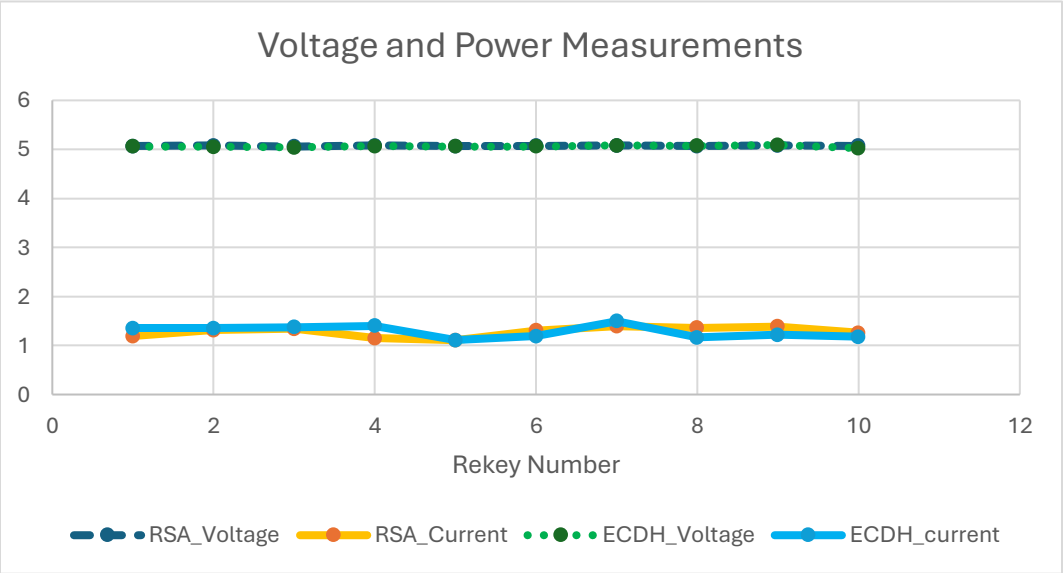
Terminal outputs also display confirmations for the use of the cryptographic accelerator on the Raspberry PI as well as periodic rekeying.

```
    Finished `dev` profile [unoptimized + debuginfo] target(s) in
0.13s
     Running `target/debug/rpi-secure-stream --mode=sender --
payload=video --host '10.42.03:5000' --device libcamera --width 1280 -
-height 720 --fps 30 --rekey 10s --metrics-dir ./metrics/sender`
ARMv8 Crypto Extensions — AES: true, PMULL: true
[app] mode=sender payload=video rekey=Some(10s)
[app] using libcamerasrc (Pi CSI camera)
[send] handshake OK
...

...
[send] rekey sent+applied
[send] rekey sent+applied
[send] rekey sent+applied
[send] rekey sent+applied
```

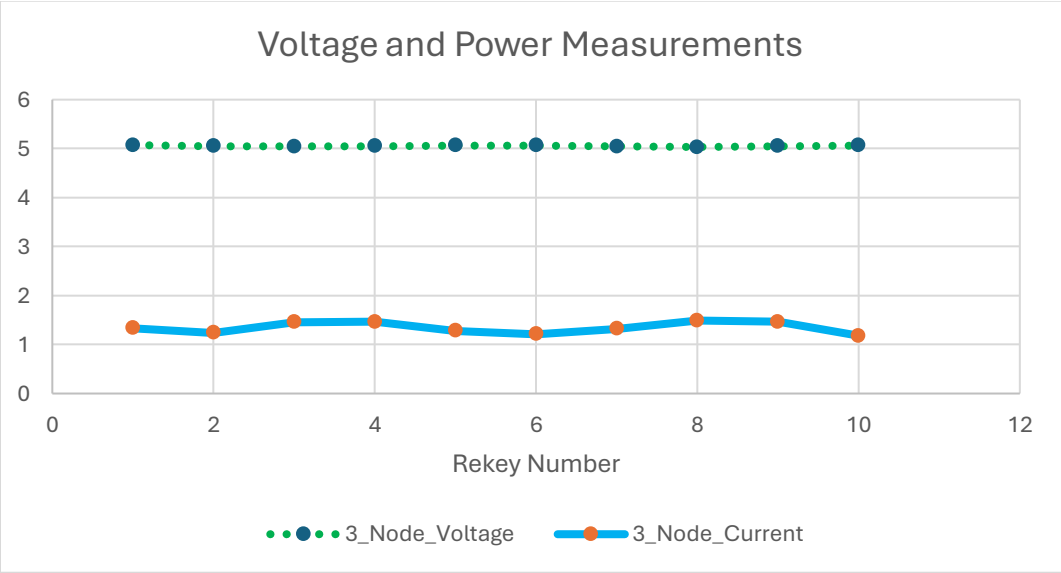
Power with Volts/Amps tables and plots for RSA vs ECDH

	RSA		ECDH	
Rekey #	V (Volts)	I (Amps)	V (Volts)	I (Amps)
1	5.065	1.189	5.057	1.357
2	5.082	1.313	5.049	1.355
3	5.057	1.337	5.041	1.373
4	5.082	1.153	5.065	1.402
5	5.065	1.108	5.057	1.112
6	5.073	1.309	5.057	1.193
7	5.082	1.394	5.078	1.494
8	5.065	1.361	5.073	1.161



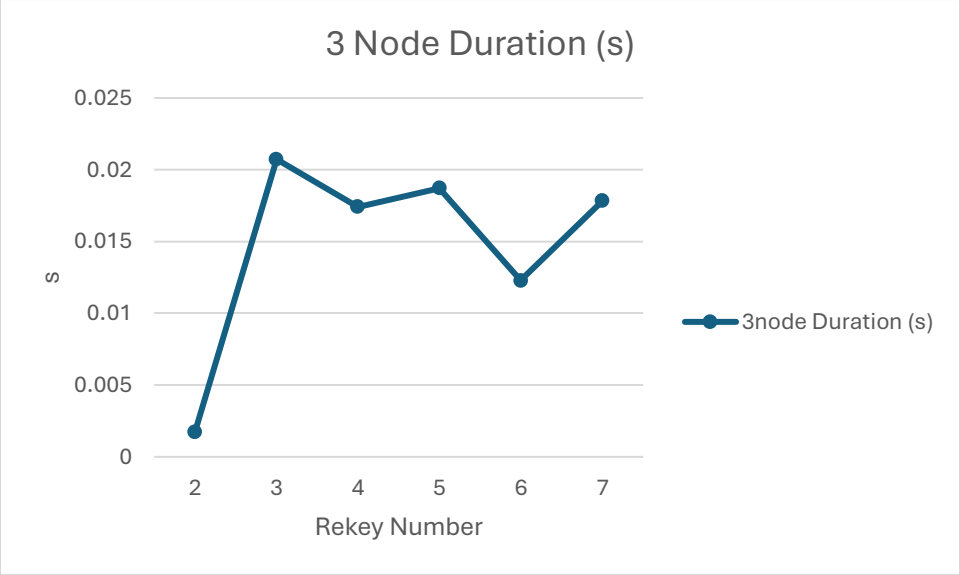
Power Measurements of 3 Node Run

	ECDH	
Rekey #	V (Volts)	I (Amps)
1	5.073	1.333
2	5.049	1.241
3	5.041	1.458
4	5.049	1.466
5	5.065	1.281
6	5.065	1.209
7	5.041	1.32
8	5.033	1.49



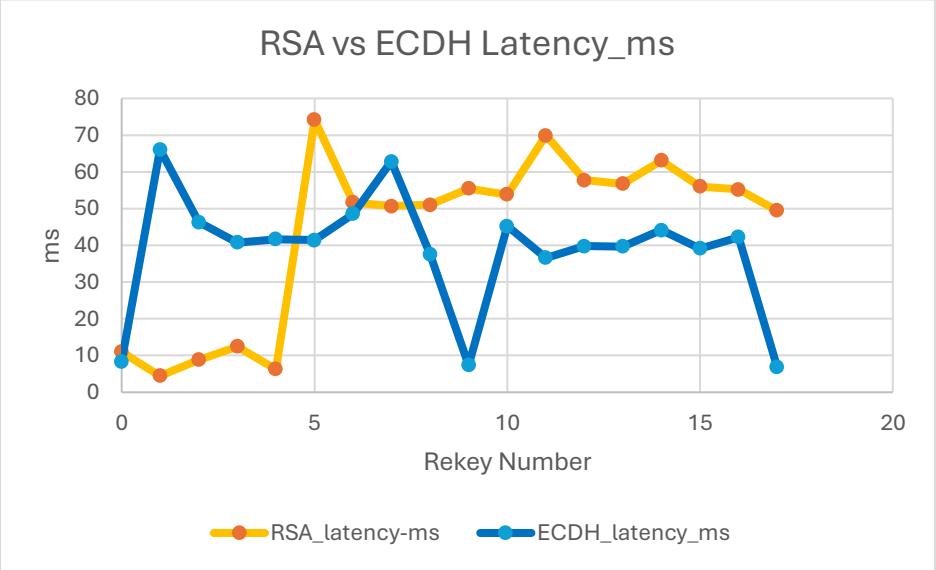
Handshake for table and plot of 3 Nodes

	3 Node				
Rekey #	timestamp	method	duration_s	bytes_exch	success
0	2025-11-0	ECDH	0.001884	75	TRUE
1	2025-11-0	ECDH	0.001737	75	TRUE
2	2025-11-0	ECDH	0.020727	75	TRUE
3	2025-11-0	ECDH	0.017426	75	TRUE
4	2025-11-0	ECDH	0.018721	75	TRUE
5	2025-11-0	ECDH	0.012279	75	TRUE
6	2025-11-0	ECDH	0.017846	75	TRUE



Latency for table and plot RSA vs ECDH

	RSA		ECDH		
Rekey #	ts	latency_ms	timestamp	latency_ms	
0	2025-11-0	10.97345	2025-11-0	8.386054	
1	2025-11-0	4.439116	2025-11-0	66.1117	
2	2025-11-0	8.862019	2025-11-0	46.31027	
3	2025-11-0	12.48789	2025-11-0	40.80957	
4	2025-11-0	6.326199	2025-11-0	41.65447	
5	2025-11-0	74.15605	2025-11-0	41.41004	
6	2025-11-0	51.62477	2025-11-0	48.60727	
7	2025-11-0	50.69757	2025-11-0	62.85501	
8	2025-11-0	51.02539	2025-11-0	37.59109	
9	2025-11-0	55.44376	2025-11-0	7.360207	
10	2025-11-0	53.84541	2025-11-0	45.20254	
11	2025-11-0	69.92173	2025-11-0	36.55739	
12	2025-11-0	57.76238	2025-11-0	39.73718	



VI. Conclusion

In summary, the project successfully integrates cryptographic rigor with real-time streaming practicality. The ECDH mechanism provides ephemeral key exchange and forward secrecy, RSA adds authentication and secure initialization, and the multi-node setup extends security to distributed group streaming. The measurements confirm that cryptographic processing introduces minimal performance impact, and the system maintains high throughput and low latency. Through detailed implementation and observation, this work shows how modern cryptographic principles can be realized in efficient, transparent code. The experience also reinforces a broader understanding of secure systems — that security is not merely about strong algorithms but about managing state, trust, and performance coherently throughout the system's lifetime. This project stands as both a demonstration and a reflection on the process of designing, implementing, and evaluating secure communication systems.