Real-Time Secure Walkie-Talkie System

By: Shawn Keelin, Louis Turriaga, Matthew West, Parsa Rezaei

Introduction

Objective

Our primary goal was to create a reliable and secure audio streaming system capable of delivering real-time communication with minimal latency.

Importance

Ensuring secure communication is paramount in various industries, including law enforcement, corporate environments, and emergency services. Our system addresses the need for protected audio transmission, safeguarding sensitive information from unauthorized access.

Original Plan

Encryption

AES-GCM for confidentiality and integrity.

Protocols

WebRTC for real-time streaming.

Security

SHA-256 for integrity and RSA for signatures.

Interface

Push-to-talk for simplicity.

Final Implementation

Encryption Shift

We transitioned from AES to ChaCha20 for encryption, recognizing its superior performance characteristics, particularly in low-resource environments.

Key Exchange

The initial RSA-based key exchange was replaced with Diffie-Hellman, a more efficient and dynamic approach for establishing secure communication keys.

Simplified Protocol

The intricate WebRTC protocol was streamlined to a simpler UDP-based solution, optimizing for real-time communication and reducing complexity.

User Interface

A toggle mode was implemented, replacing the traditional push-to-talk interface, offering more seamless and intuitive communication.

System Architecture

1

Key Exchange (Diffie-Hellman)

Diffie-Hellman securely generates a shared key by exchanging public keys, allowing both devices to compute the same key without transmitting sensitive information.

2

Creation and Exchange of RSA Keys

Each device generates an RSA key pair (public and private keys) for signing and verifying messages, exchanging public keys securely to ensure authenticity.

3

Audio Capture

The system begins with audio capture, where the microphone on the transmitting device captures the user's voice signal.

4

Encryption (ChaCha20)

The captured audio data is then encrypted using ChaCha20, a robust and efficient encryption algorithm, and the shared key created in the first step.

5

Signing (RSA)

Following encryption, the data is signed using RSA, ensuring authenticity and integrity, effectively preventing tampering and verifying the sender.

6

UDP Transmission

The encrypted and signed audio data is packaged into UDP packets and transmitted over the network, leveraging the efficiency and low overhead of UDP for real-time communication.

7

Decryption and Playback

On the receiving device, the encrypted data is decrypted using ChaCha20 and the RSA signature is verified to ensure data authenticity. The audio is then played back, delivering the intended message.

Demo Setup

1. **Receive**

   Start the demo by acting as the receiver, ready to receive encrypted audio transmissions.

2. **Transmit**

   Use the transmitting device to send encrypted audio messages, showcasing the real-time communication functionality.

3. **Observe**

   Observe the real-time logging interface to monitor the data flow, encryption and decryption processes, and other relevant information.

4. **Toggle**

   Demonstrate the toggle mode functionality, highlighting its ease of use and efficiency for seamless communication.

```python
# Global Variables
current_process = None  # Current subprocess
is_transmitter = False  # Mode flag
target_ip = None  # Target IP address
UDP_PORT_RX = None  # UDP port for receiving
UDP_PORT_TX = None  # UDP port for transmitting
shared_key = None  # Shared encryption key

def stop_current_process():
    global current_process
    if current_process:
        try:
            print("Stopping current process...")
            current_process.terminate()
            current_process.wait()
            current_process = None
        except Exception as e:
            print(f"Error stopping process: {e}")

def ensure_metrics_file():
    if not os.path.exists("metrics.log"):
        with open("metrics.log", "w") as f:
            f.write("Timestamp,Message\n")

def log_metric(message):
    with open("metrics.log", "a") as f:
        f.write(f"{datetime.now()}: {message}\n")

def generate_rsa_keys():
    if not os.path.exists("private_key.pem") or not os.path.exi
        start_time = time.time()
        try:
            private_key = rsa.generate_private_key(
                public_exponent=65537,
                key_size=2048,
                backend=default_backend()
            )
            public_key = private_key
            with open("private_key.p                          le:
                private_file.write(private_key.private_bytes(
```

Performance Metrics: Encryption Time Comparison

~2.2ms

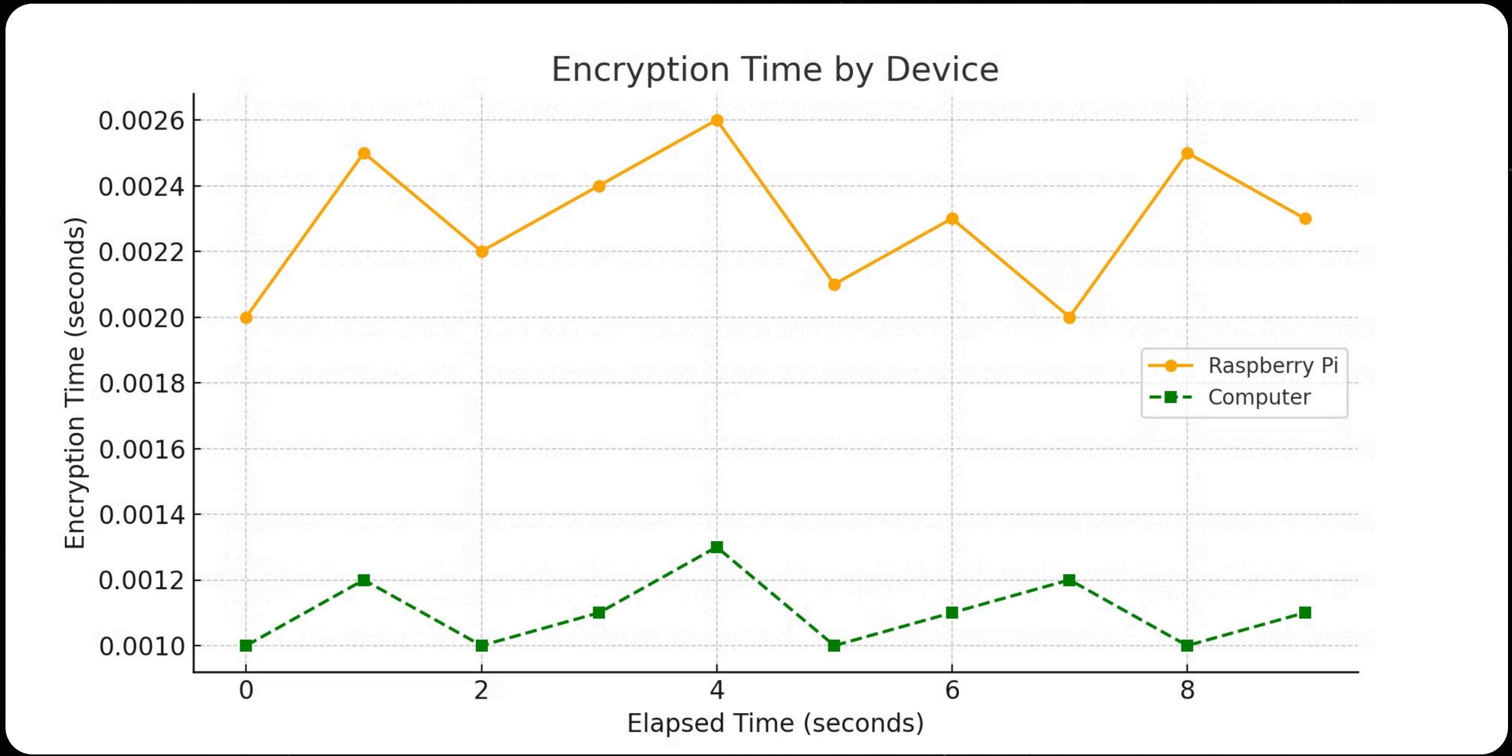
Raspberry Pi

Slower due to limited processing power.

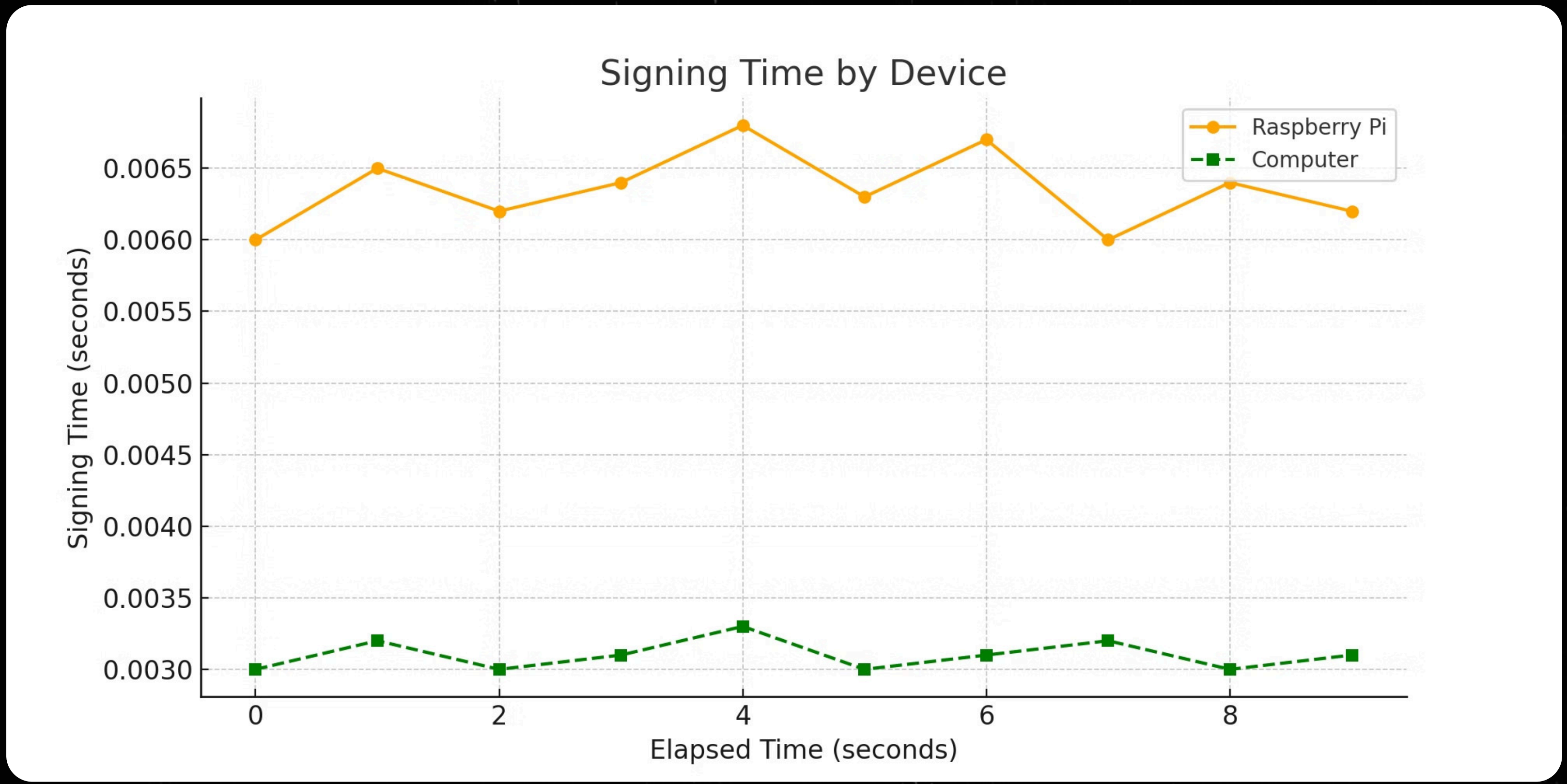
~1.1ms

Computer

Faster processor, faster encryption.



Performance Metrics: Signing Time Comparison



~6.3ms

Raspberry Pi

~3.2ms

Computer

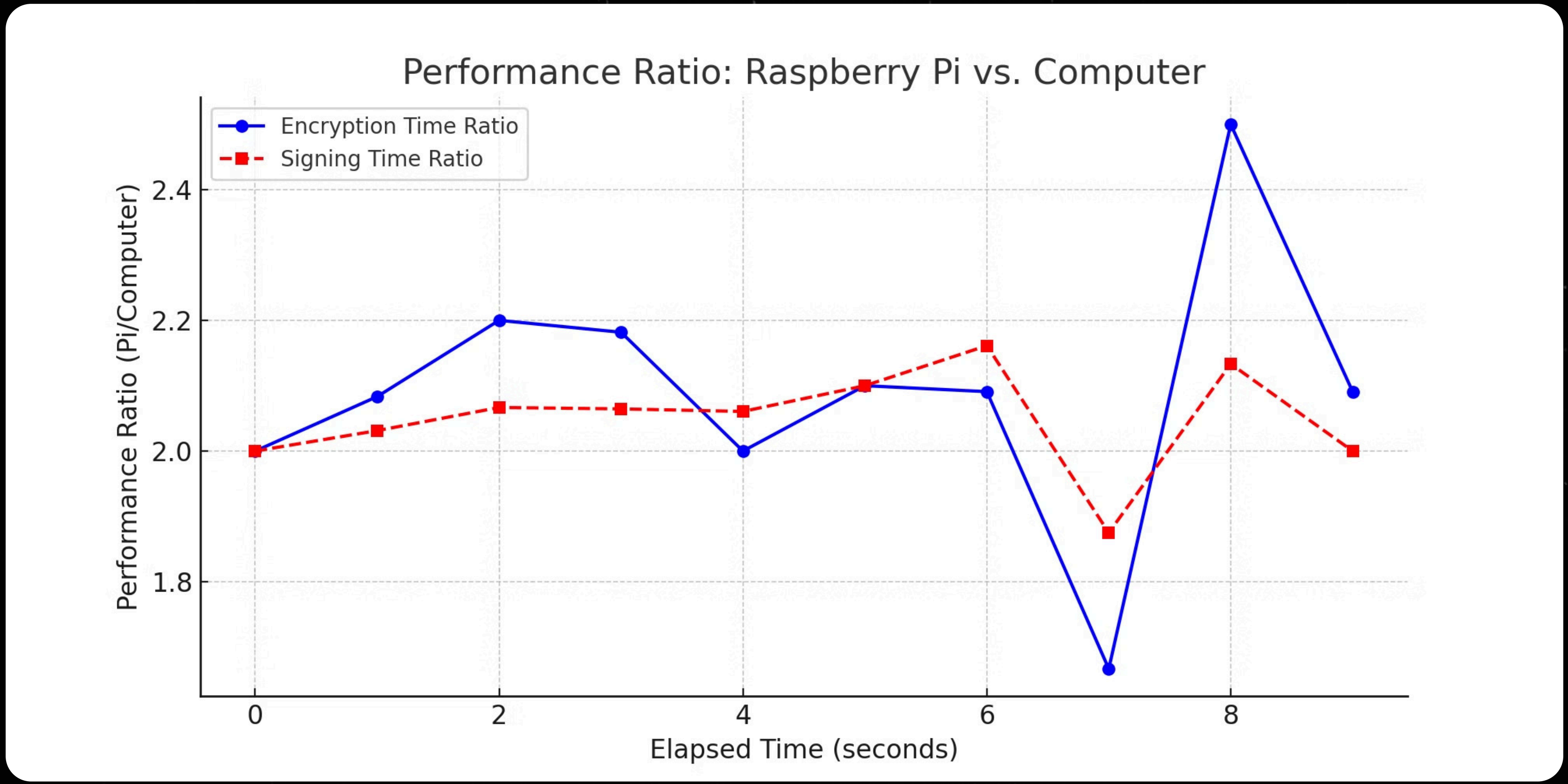
Raspberry Pi

The Raspberry Pi's signing process takes approximately twice as long as the computer, indicating potential latency in real-time communication.

Computer

The computer consistently demonstrates faster and more stable signing performance, enabling efficient communication that is twice as fast as the Raspberry Pi.

Performance Ratio: Raspberry Pi vs. Computer



Encryption

Raspberry Pi is ~2x slower.

Signing

Raspberry Pi is ~2x slower.

Challenges and Lessons Learned

1

Latency

Managing latency, particularly on the Raspberry Pi, was a critical challenge, requiring optimization techniques and careful consideration of resource constraints.

2

Trade-offs

Balancing performance and security was an ongoing challenge, demanding careful evaluation and selection of cryptographic algorithms and system design choices.

3

Reliability

Ensuring network reliability was a crucial factor, influencing the overall performance and stability of the real-time communication system.

4

Logging

The value of detailed logging became apparent, providing valuable insights into system behavior, performance, and potential issues.

Questions?