NETWORK SECURITY PROJECT REPORT: PEER-TO-PEER MESSAGING APPLICATION

Jairik McCauley, Logan Kelsch, & Aaron Triplett

I. Abstract

This project details the creation of a secure peer-to-peer (P2P) messaging application that leverages advanced cryptographic methods and networking protocols. Utilizing UDP multicast for efficient peer discovery, ChaCha20-Poly1305 for robust encryption, and Ed25519 digital signatures for authentication, the application addresses some contemporary challenges in network security, those being confidentiality and authenticity. The implementation utilizes Python's cryptography library, providing a practical demonstration of secure communication practices suitable for diverse real-world applications, such as collaborative environments.

II. Introduction

Network security remains one of the most critical issues in digital information transfer and management, exacerbated by growing decentralization, increased reliance on wireless or interceptable networks, and threats growing in sophistication. Mobile devices are a prime example, where everything, sensitive and non-sensitive, is transmitted wirelessly. From bank and payment information to health documents and text messages, all of this information can be put at risk if not encrypted, as it is transmitted over an interceptable medium. To demonstrate a solution to these concerns comprehensively, this project combines state-of-the-art cryptographic protocols—ChaCha20-Poly1305 [1] and Ed25519 [4]—integrated with UDP multicast technology [7], a minimal dependency graphical user interface (GUI), dearpygui, and simple multi-threading. This approach provides a balanced solution that ensures secure, authenticated, and fast communication while supporting dynamic peer discovery.

III. Cryptographic Preliminaries

ChaCha20-Poly1305

ChaCha20-Poly1305 [1] represents an advanced cryptographic algorithm combining encryption and authentication. ChaCha20 is a stream cipher designed for high performance and resistance to side-channel attacks, making it particularly suited for network protocols requiring low latency and high throughput. Poly1305 serves as a message authentication code (MAC), ensuring message integrity and preventing unauthorized modifications during transmission. This combination, standardized in RFC 8439, offers superior performance compared to earlier standards such as AES-GCM, especially on devices with limited computational resources. Procter's security analysis [2] and subsequent research by Degabriele et al. [3] extensively validated this algorithm, providing rigorous theoretical and practical assurances of its security in both single-user and multi-user environments, crucial for public networks.

Ed25519 Digital Signatures

Digital signatures are vital in securing network communications by providing verifiable authenticity and non-repudiation—key aspects in combating impersonation and forgery attacks. Ed25519 [4], introduced by Bernstein et al., provides these capabilities with a notable balance between security and efficiency [5, 6]. It is optimized for speed without compromising security, making it ideal for real-time communication and resource-constrained environments like IoT devices. Standardized by RFC 8032, Ed25519 ensures interoperability across various platforms, contributing significantly to network security protocols.

IV. Methods

The technical solution presented involves developing a P2P communication system that integrates sophisticated cryptographic functions and efficient networking methods through three essential Python modules:

Client Application

The client application serves as the user's window into the P2P network, managing user interface, networking, and cryptographic duties in concert. On launch, it initializes a DearPyGui viewport for username entry and, once a valid name is submitted, smoothly transitions to a messaging window—resizing the UI and displaying the user's handle. Underneath the GUI, two threads begin running: one continuously listens for incoming multicast packets on a UDP socket (created with 'SO_REUSEADDR' and joined to the multicast

group via 'IP_ADD_MEMBERSHIP'), while the other periodically broadcasts "JOIN" messages containing the encrypted username and Ed25519 public key.

Whenever the user types a message and clicks "Send," the application packages the text with the ChaCha20-Poly1305 cipher—ensuring confidentiality—and signs the plaintext with the Ed25519 private key—ensuring authenticity. These three elements (ciphertext, nonce, signature) are then pickled into a "CHAT" packet and sent to the multicast group. Incoming packets are unpickled in the listener thread: "CHAT" packets are decrypted, signatures are verified against the sender's stored public key, and valid messages are logged in the GUI's scrolling message window. "JOIN" and "LEAVE" packets similarly update the local peer registry, allowing the application to display join/leave notifications in real time. Robust error handling throughout socket operations, thread shutdown, and GUI callbacks guarantees that the client can recover gracefully from network hiccups, invalid input, or forced exits—providing a seamless, secure messaging experience.

Configuration Module

The `config.py` file centralizes all protocol-level constants and cryptographic parameters, keeping them decoupled from application logic. By defining these values in one place, we simplify maintenance, testing, and future reconfiguration.

- **Multicast Settings:**
  - `MCAST_PORT = 5000` Specifies the UDP port on which all clients bind and listen for multicast traffic. For this project, a well-known, fixed port allows peers to interoperate without manual configuration.
  - `MCAST_GRP = '224.1.1.1'` Defines the IPv4 multicast group address used for peer discovery and message exchange.
- **Networking Parameters:**
  - `BUFFER_SIZE = 4096` Sets the maximum datagram size (4KB) the socket will read in a single `recvfrom()` call. This accommodates typical P2P payloads—including ciphertext, nonces, and signatures—while limiting memory overhead.
- **ChaCha20-Poly1305 Settings:**
  - `KEY_SIZE = 32` Defines the required key length (in bytes) for the ChaCha20 stream cipher.

- `NONCE_SIZE = 12` Specifies the 96-bit nonce length for ChaCha20-Poly1305, preventing nonce reuse and ensuring strong cryptographic guarantees.
- `SHARED_CHACHA20_KEY = b'\x01' * 32` Holds the 256-bit symmetric key pre-shared among clients. At a larger scale, this value would be distributed via a secure key-exchange mechanism rather than hard-coded.

- **Protocol Helpers:**
  - `NULL_BYTE = b'\x00'` A literal zero byte used as a placeholder for unused fields (e.g., signature or public-key slots) in certain packet types, ensuring consistent packet structure across all message variants.

By grouping these settings in `config.py`, the application achieves clear separation of concerns: network parameters, cryptographic constants, and buffer limits that can be tuned independently of the core messaging logic, facilitating easier testing and future enhancements.

Cryptographic Utility Module

The crypto utilities module serves as the dedicated security engine for the client, abstracting away the complexity of encryption, decryption, and signing behind a concise interface. Upon loading, it immediately instantiates a 'ChaCha20Poly1305' object with the shared symmetric key defined in the Configuration Module, ensuring that every outgoing message benefits from authenticated encryption. Simultaneously, it generates a fresh Ed25519 private key, granting each client a unique signing identity.

When the application needs to send data, it calls the single function 'pack_data()', which performs four critical operations in one go: it either accepts or generates a cryptographically secure random nonce; it encrypts the plaintext with ChaCha20-Poly1305 (automatically binding any associated data into the authentication tag); it optionally signs the original plaintext using Ed25519 to produce a verifiable signature; and finally, it returns a neatly packaged tuple— '(ciphertext, nonce, signature)' or '(ciphertext, nonce)' when signing is skipped. By centralizing these steps, the module prevents common pitfalls such as nonce reuse and omitted authentication.

On the receiving side, 'unpack_data()' accepts the ciphertext and its corresponding nonce (and any associated data), decrypts and verifies the Poly1305 authentication tag in one atomic operation, and catches any failures—be they tampered payloads or authentication mismatches—returning 'None' to signal a bad

packet. Successful decryptions yield UTF-8 decoded strings for chat messages or raw bytes when handling key material, ready for higher-level logic to display.

Two helper routines round out the API: 'get_ed_public_key()' serializes the client's public key into a compact raw-bytes format suitable for broadcast in "JOIN" handshakes, and 'verify_signature()' reconstructs an Ed25519 public key from raw bytes to authenticate incoming messages. This narrative-driven design ensures that all cryptographic concerns—key management, nonce generation, AEAD encryption, and signature verification—are centralized and auditable, yielding a secure, maintainable foundation for end-to-end message confidentiality and authenticity.

V. Results

Practical evaluations demonstrated the system's capacity to securely manage real-time communications. Key findings include:

- Rapid and reliable peer discovery using UDP multicast, with peers becoming seamlessly integrated into the communication network.
- Efficient cryptographic operations, enabling low-latency encrypted messaging suitable for live communications and interactions, while robustly ensuring message authenticity and integrity across the network with minimal computational overhead.

These results affirm the practical feasibility of applying advanced cryptographic techniques in a peer-to-peer environment.

VI. Security Analysis

An extensive security assessment highlighted critical strengths and provided insights into potential vulnerabilities:

Encryption and Authentication

Rigorous testing [2] confirmed the robustness of ChaCha20-Poly1305 encryption against known cryptographic threats, effectively safeguarding confidentiality and integrity. Ed25519 signatures proved secure against forgery and replay attacks, affirming the identity and authenticity of communicating parties, essential in public and distributed network environments.

Network and Protocol Security

The multicast-based approach significantly mitigates traditional vulnerabilities associated with centralized networks. Decentralization eliminates single points of failure, reduces targeted attack surfaces, and increases overall resilience. However, it introduces complexities in securely managing cryptographic keys, requiring secure distribution and management protocols to prevent unauthorized access.

VII. Network Security Implications

The broader implications of this work on network security are substantial. Integrating robust cryptographic mechanisms within decentralized communication protocols directly addresses and mitigates prevalent network threats such as eavesdropping, message tampering, identity spoofing, and denial-of-service attacks. By establishing secure communication channels inherently resistant to these threats, this work sets foundational guidelines for future decentralized network designs and implementations.

VIII. Significance and Applications

The secure messaging framework developed here has extensive real-world applicability across various domains requiring secure decentralized communication. Potential applications include secure communication for IoT networks, collaborative platforms requiring authenticated interactions, emergency response systems leveraging decentralized infrastructure, and secure data exchange in corporate or government networks. The principles and methods described in this report provide a comprehensive template for developing secure communication solutions adaptable to diverse networking scenarios.

References

[1] Internet Research Task Force (IRTF), *ChaCha20 and Poly1305 for IETF Protocols*, RFC 8439, June 2018. https://www.rfc-editor.org/rfc/pdfrfc/rfc8439.txt.pdf

[2] G. Procter, *A Security Analysis of the Composition of ChaCha20 and Poly1305*, https://eprint.iacr.org/2014/613.pdf

[3] J. P. Degabriele, J., et al. *The Security of ChaCha20-Poly1305 in the Multi-user Setting*, https://eprint.iacr.org/2023/085.pdf

[4] Bernstein, D. J., et al. (2011). *High-Speed High-Security Signatures (Ed25519)*, https://ed25519.cr.yp.to/ed25519-20110926.pdf

[5] Internet Research Task Force (IRTF), *Edwards-Curve Digital Signature Algorithm (EdDSA)*, RFC 8032 (2017). https://datatracker.ietf.org/doc/html/rfc8032

[6] Owens, et al. (2023). *Effcient and Side-Channel Resistant Ed25519 on ARM Cortex-M4*, https://par.nsf.gov/servlets/purl/10507285

[7] The 8472 (2015). *BitTorrent Local Service Discovery (UDP Multicast)*, https://www.bittorrent.org/beps/bep$_0$014.*html*