

Cipher: An End-to-End Encrypted Peer-to-Peer Instant Mobile Messaging System

Miraç DEPREM

Department of Computer Engineering, Faculty of Technology, Isparta University of Applied Sciences, Isparta, Türkiye
Email:mrddprmm@gmail.com

Eren Ali KOCA

Department of Computer Engineering, Faculty of Technology, Isparta University of Applied Sciences, Isparta, Türkiye
Email:eren_ali_koca@hotmail.com

Ahmet Ali SÜZEN

Department of Computer Engineering, Faculty of Technology, Isparta University of Applied Sciences, Isparta, Türkiye
Email:ahmetsuzen@isparta.edu.tr

ABSTRACT

Most instant messaging applications rely on a centralized client-server architecture, which entails systemic risks such as data privacy concerns, censorship, and single points of failure. This study designs and implements Cipher, a peer-to-peer mobile instant messaging system that communicates without transmitting message content to a central server in plain text. The network and cryptography core was developed in Go using hexagonal architecture; Android and iOS clients were built with Flutter and integrated via the Gomobile bridge. Security employs RSA-4096 identity keys, HMAC-SHA256-based deterministic session key derivation, and AES-256-GCM authenticated encryption. A composite transport layer combines UDP, a WebRTC data channel, ICE with STUN/TURN, and an encrypted Firebase relay fallback. Field tests showed a connection success rate above 95%, an average 30% latency increase in relay mode versus direct P2P, reliable large-file transfer over UDP through fragmentation and rate limiting, and application startup reduced from 5 s to under 1 s. The findings indicate that a mobile-first, secure, and usable decentralized communication prototype is feasible with appropriate architectural choices.

Keywords—End-to-end encryption, Mobile messaging, NAT traversal, Peer-to-peer communication, WebRTC.

Date of Submission: June 16,2026

Date of Acceptance: August 10,2026

I. INTRODUCTION

Today's digital communication infrastructure relies heavily on a centralized client-server model. Popular instant messaging apps like WhatsApp and Telegram manage all user data and communication flows through servers controlled by a single authority. While this architecture offers operational convenience, it inherently carries a number of systemic risks. The centralization of user data creates a "honeypot effect," making this data an attractive target for attackers; it also undermines data sovereignty by allowing the service provider to analyze, profile, and share the data with third parties [1]. Furthermore, centralized servers can be easily monitored and censored; the regional blocking of a service is a natural consequence of this model.

Another vulnerability of centralized architecture is the risk of a single point of failure. In October 2021, a global outage affecting Facebook's services—caused by a faulty BGP routing configuration and affecting billions of users for hours—concretely demonstrated the scale of this risk. Peer-to-peer networks, a fundamental alternative to these systemic issues, distribute tasks and workloads directly among peers without a central intermediary; because they lack a single point of failure, they offer a more resilient, censorship-resistant, and data-sovereign model [2].

It is sometimes argued that security and privacy are already implemented on the server side, and that additional evidence for an end-to-end design is therefore unnecessary. Server-side controls (TLS in transit, disk encryption at rest, and access policies) do not prevent the operator from reading plaintext, profiling users, or complying with compelled disclosure [1], [3]. Even applications that advertise end-to-end encryption typically retain metadata—who communicated with whom, when, and from which network—on centralized servers [3]. Telegram's default cloud chats are stored on provider servers rather than being end-to-end encrypted; only optional Secret Chats provide device-to-device encryption [3]. Forensic studies of Android messengers further show that once genuine end-to-end encryption is in place, investigators cannot recover message plaintext from company servers and must instead target device-side caches [4]. These observations provide concrete evidence that server-side security does not substitute for end-to-end encryption held on user devices.

The aim of this study is to design, implement, and evaluate through field tests the Cipher prototype, which integrates the above concepts into a mobile-first and engineering-feasible system. Cipher—comprising a high-performance Go-based networking and cryptography core, a Flutter

mobile interface, a multi-transport protocol, and a hybrid discovery-relay architecture—aims to enable secure communication without transmitting plaintext messages to a central server. Rather than proposing a novel cryptographic primitive, this work contributes an engineering-oriented architecture that integrates proven cryptographic and networking techniques into a practical mobile-first decentralized messaging platform, which is then evaluated experimentally through cross-platform field tests.

The remainder of this paper is organized as follows. Section 2 reviews related literature. Section 3 presents the materials and method. Section 4 reports the field-test results and a comparison with existing methods. Section 5 discusses the findings. Section 6 concludes.

II. LITERATURE REVIEW

P2P networks are classified into three main categories based on the structure of the overlay network: unstructured, structured, and hybrid [5]. In unstructured networks, connections are established randomly, and resource discovery is typically performed via flood; while they are resilient to churn, they have low scalability. Structured networks use Distributed Hash Tables such as Kademlia [6], Chord [7], Pastry [8], and CAN [9] to locate resources in a deterministic and efficient manner, but maintaining the topology imposes an additional management burden. In mobile environments, the requirement for pure P2P discovery mechanisms to maintain constant background network activity drives battery and data consumption to unacceptable levels. For this reason, in practical mobile systems, hybrid architectures supported by central components are preferred for specific tasks such as peer discovery and signaling.

End-to-end encryption, which ensures that data can only be read on the sender's and recipient's devices, forms the foundation of secure communication [3]. The Signal Protocol, which has become the de facto standard in the industry, provides advanced features such as forward secrecy and post-compromise security through the Extended Triple Diffie-Hellman key exchange [10] and the Double Ratchet algorithm; formal security analyses of the protocol have also demonstrated that it offers strong guarantees [11], [12]. The Network Address Translation (NAT) problem—the most common practical obstacle to P2P communication—is overcome using the gradual degradation principle of the STUN, TURN, and ICE protocols [13].

Surveys of secure and decentralized messengers show that deployed systems occupy different points in the design space of confidentiality, usability, and degree of decentralization [1], [14]. Centralized applications such as WhatsApp and Signal provide strong end-to-end encryption based on the Signal Protocol, yet they route message delivery and metadata through centrally operated

servers, retaining a single point of control and failure [1], [3]. Telegram's default cloud chats are stored on provider servers; only optional Secret Chats are end-to-end encrypted [3], [14]. Fully decentralized, DHT-based clients such as Tox/qTox carry messages without a central store, but NAT traversal and always-on mobile operation remain difficult [14]. Briar relies on Tor and direct Bluetooth/Wi-Fi links to remove central infrastructure, at the cost of higher latency and reduced usability on always-connected mobile devices. Session replaces long-term identifiers with an onion-routing overlay, whereas Matrix adopts a federated model in which independent homeservers synchronize conversations [1], [14]. Related P2P systems such as Tribler illustrate social-based overlay designs outside the instant-messaging setting [15]. Cipher differs from these systems by combining direct peer-to-peer transport with a lightweight centralized layer used only for discovery and encrypted relay, deliberately trading a degree of decentralization for practical mobile usability while keeping message plaintext off the server.

Authenticated encryption with AES-GCM is a well-established construction for simultaneous confidentiality and integrity. A recent experimental comparison of AES, 3DES, and Blowfish published in this journal found AES to offer the strongest resistance to attacks while remaining competitive in throughput, which supports Cipher's choice of AES-256-GCM on mobile devices [16].

III. MATERIALS AND METHOD

Cipher was developed as a system consisting of two main components: the network and cryptography core (Cipher), written in the Go language, and the Flutter-based cross-platform mobile client (Cipher-Mobile). The core was compiled using Gomobile and embedded into the mobile client as a native library. This section provides a detailed explanation of the system's architecture, network protocols, cryptographic mechanisms, mobile application layer, and testing methodology.

3.1 System Architecture

Cipher's core has been developed using a hexagonal architecture in accordance with the principles of Clean Architecture and Domain-Driven Design. The core consists of five layers: the domain layer contains pure business rules and value objects; the ports layer contains interface contracts that enable dependency inversion; the use case layer contains application workflows; the adapters layer handles communication with the outside world; and the platform layer contains concrete implementations of infrastructure such as UDP, WebRTC, cryptography, and storage. Thanks to this separation, changing the transport protocol or storage technology does not affect the core business logic.

Fig. 1 summarizes the system's overall architecture. For mobile deployment, the Go core is compiled using Gomobile into .aar files for Android and .xcframework

files for iOS; on the Flutter side, it is managed via MethodChannel and EventChannel bridges. Communication between the user interface and the Go core is asynchronous, and computationally intensive cryptographic operations are executed in the background on the Go side, thereby maintaining interface responsiveness.

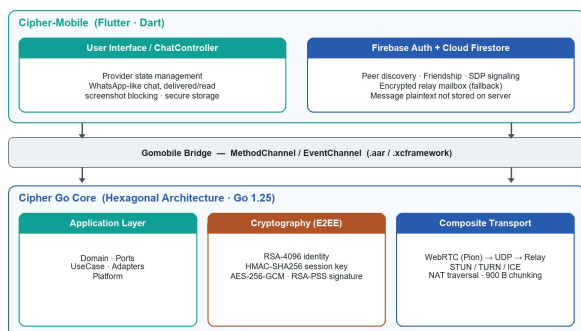


Fig. 1: Cipher system architecture and inter-layer communication

3.2 Back-end and network protocols

The system adopts a composite transport approach that dynamically adapts to network conditions. The primary transport is a WebRTC data channel that provides NAT traversal and ICE support, and is implemented using Pion, a mature library in the Go ecosystem; when a WebRTC connection cannot be established, low-latency UDP transport takes over. Both transport mechanisms use the same JSON frame structure and carry only an encrypted byte sequence in the payload field. Table 1 compares the transport protocols used.

TABLE 1: COMPARISON OF TRANSPORT PROTOCOLS

Protocol	Usage	Advantage	Limitation
WebRTC DataChannel	Primary P2P	NAT/ICE support	Signaling required
UDP	Fallback direct communication	Low latency	NAT/firewall sensitivity
Firebase Relay	Offline / fallback	High availability	Centralized component, additional latency

When the UDP listener is started, it sends a binding request to Google STUN servers to discover its XOR-mapped public address and updates the local listening address metadata; this information is reflected in the peer discovery directory. On the WebRTC side, a data channel named “cipher” is created; the SDP offer and response are generated on the Go side, and ICE candidate exchange and signaling are handled via Firestore. STUN and TURN configuration is passed from the client to the core as JSON at runtime.

Firebase Authentication and Cloud Firestore are used for peer discovery, friendship management, and WebRTC signaling. Message content is never stored in plain text on the server. When direct P2P communication fails or the recipient is offline, the encrypted relay frame generated on the Go side is written to Firestore according to an inbox

pattern and decrypted when the recipient comes online; processed frames are automatically cleared. To simplify the user experience, a 16-character user-friendly identifier (Simple ID)—derived from the SHA-256 hash of the public key—is used instead of the raw public key.

3.3 File Transfer and Flow Control

Large files are transferred by being split into chunks at the application layer using the FILE_START, FILE_CHUNK, and FILE_END media types. Taking into account UDP MTU limits and the JSON frame overhead, the chunk size was set to 900 bytes; this ensures that the total packet size remains below the safe WAN limit of 1024 bytes. Each chunk is encoded with a compact binary header containing the version, transfer ID, sequence number, and total number of chunks. To prevent buffer overflow caused by UDP’s lack of flow control, a pacing delay of 2–5 ms is inserted between chunks. On the receiving end, the fragments are reassembled into a temporary file; after FILE_END is received, a timeout is observed for missing fragments, and the size of the completed file is verified against the expected value.

3.4 Encryption mechanisms

In Cipher, E2EE operations are performed not in the Dart layer, but in the Go secure cryptography package. Each device generates an RSA-4096 [17] key pair. The peer ID is the hexadecimal representation of the first 8 bytes of the SHA-256 hash of the public key’s PKIX DER encoding. The private key is stored in PKCS#1 format, and the public key in PKIX DER format; the RSA-PSS scheme is used for signing operations.

A deterministic session key is derived using HMAC-SHA256 based on the public key hashes of both peers and their sequential peer IDs. This approach allows both parties to generate the same key independently and without an additional handshake round; however, the forward secrecy provided by the X3DH and Double Ratchet mechanisms in the Signal Protocol has intentionally been left for future work. The first 32 bytes of the session key are used for AES-256-GCM [18]; a random nonce is generated for each message and appended to the ciphertext. Since GCM is an authenticated encryption mode, successful decryption is considered practical verification that the message originated from the authorized sender. In group messaging, instead of a shared group key, a fan-out model is used, in which a separate, one-to-one encrypted message is sent to each member.

3.5 Mobile application layer

Cipher-Mobile is organized using a feature-first architecture. The central ChatController component coordinates messaging, peer management, P2P pre-warming, file transfer, presence status, and the relay inbox. The reactive Provider pattern is used for state management. The user interface has been localized into Turkish; the conversation screen features WhatsApp-style

message bubbles, sent/read indicators, and file previews (Fig. 2). Security measures include screen capture prevention, hardware-based secure storage of identity keys, and session termination when the app is minimized.

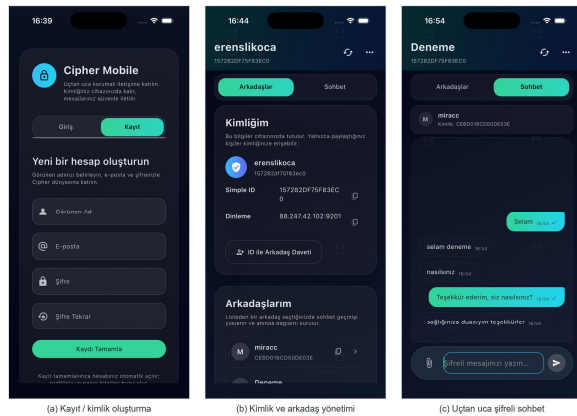


Fig. 2: Cipher-Mobile interface: (a) registration, (b) account/friends management, (c) end-to-end encrypted chat

3.6 Testing and Evaluation Methodology

As part of quality assurance, a multi-layered testing strategy was followed: unit tests were run on the Go side, widget tests on the Flutter side, and two-node HTTP+UDP integration tests; external dependencies were isolated using mock objects. Network performance was evaluated through cross-platform field tests on Android and iOS devices; NAT type, connection mode (direct P2P, TURN, relay), latency, file integrity, and app launch time were measured. On the deployment side, the Go code was cross-compiled into native libraries for Android and iOS, and production output was automated using continuous integration (CI) pipelines.

IV. RESULTS

This section presents the main findings obtained during the development and field testing of the Cipher prototype. A summary of the performance metrics for these findings is provided in Table 2. To place these metrics in context, Section 4.5 compares Cipher with existing messaging systems.

TABLE 2: SUMMARY OF FIELD TEST FINDINGS

Metric	Result
Connection success rate (composite transport)	>95%
Average latency increase in relay mode	≈30%
Application startup time (post-optimization)	<1 s
UDP file chunk size	900 bytes
Heartbeat / offline timeout	10 s / 30 s

4.1 Connection Setup and NAT Bypass

Field tests have shown that the success of direct P2P connections depends largely on the type of NAT. Measurements were obtained from 120 connection attempts across 8 Android and iOS devices under different NAT configurations. It was found that using STUN alone was insufficient for establishing connections on devices

behind symmetric NAT; TURN or encrypted relay fallback was required [19]. It was measured that the connection success rate was increased to over 95% thanks to the WebRTC-priority, UDP-based, relay-backed composite transport layer. In relay mode, however, the average latency was found to increase by approximately 30% compared to direct P2P mode. This result confirms that P2P priority and proper ICE configuration are critical for real-time messaging.

4.2 Reliable file transfer over UDP

In the initial tests of transferring large files over UDP, it was observed that packet loss occurred due to the recipient device's network buffer filling up, resulting in corrupted files. This issue was resolved at the application layer using 900-byte segmentation and a rate-limiting mechanism of 2–5 ms; file transfer was made reliable through reassembly logic that detects missing segments and size verification. Message encryption and decryption operations were completed in the order of milliseconds; it was observed that the Gomobile bridge did not cause any noticeable lag (jank) in the user interface.

4.3 Startup Performance and Concurrency

Initially, when the node was started, the concurrent presence service and profile manager attempted to access the same lock (mutex) resource, which blocked the main thread and caused a timeout error on the Flutter side. By converting the startup routines from a synchronous structure to a non-blocking goroutine structure, this deadlock was resolved, and the app's startup time was reduced from approximately 5 seconds to less than 1 second. This finding demonstrates that resource management on mobile devices is much more sensitive than in server environments.

4.4 Distributed State Management

Tracking the “delivered” and “read” statuses without a central database was addressed using acknowledgment (ACK) packets at the application layer. Status feedback was transmitted in the opposite direction using the ACK_DELIVERED and ACK_READ media types, corresponding to the unique identifier assigned to each message. Tests showed that the impact of this additional traffic on bandwidth was negligible, while it significantly improved the user experience. A 10-second heartbeat signal and a 30-second offline timeout were used for presence status.

4.5 Comparison with Existing Methods

Table 3 compares Cipher with representative existing methods along architectural and security dimensions. Cipher's quantitative rows (connection success above 95%, about 30% extra latency in relay mode, startup under 1 s, and 900-byte UDP chunks) are taken from the field tests in Table 2. Values for WhatsApp, Signal, Telegram, Tox, Briar, Session, and Matrix are qualitative properties reported in the literature [1], [3], [11], [14], [15]; they are not same-laboratory latency or throughput measurements.

The table is intended to give readers a clearer picture of where Cipher sits relative to centralized, federated, and fully decentralized messengers.

TABLE 3: COMPARISON OF CIPHER WITH EXISTING MESSAGING SYSTEMS

System	Architecture	Default E2EE	Forward secrecy	Content on server
Cipher	Hybrid P2P + encrypted relay	Yes (AES-256-GCM)	No (deterministic session key)	Ciphertext only (relay fallback)
WhatsApp	Centralized	Yes (Signal Protocol)	Yes	No (metadata retained)
Signal	Centralized	Yes (Signal Protocol)	Yes	No (minimal metadata)
Telegram	Centralized	No (optional Secret Chats)	Secret Chats only	Yes (cloud chats)
Tox / qTox	Pure P2P (DHT)	Yes	Limited	None
Briar	P2P (Tor / Bluetooth)	Yes	Yes	None
Session	Onion-routing overlay	Yes	Limited vs Signal	Onion nodes (not operator inbox)
Matrix	Federated homeservers	Optional (Megolm)	Partial	Homeserver stores history

Relative to WhatsApp and Signal, Cipher keeps message plaintext off any operator server and prefers direct P2P paths, at the cost of missing Signal-level forward secrecy and of a residual encrypted relay. Relative to Tox and Briar, Cipher accepts a small centralized discovery/signaling plane in order to raise mobile connection success above 95% under mixed NAT. Relative to Telegram, Cipher does not store cloud-chat plaintext. A same-laboratory quantitative benchmark of latency and energy against these systems remains future work.

V. DISCUSSION

The findings demonstrate that a mobile-first communication system—which operates without transmitting message plaintext to a central server—is feasible in practice. The fact that the composite transport layer achieves a connection success rate of over 95% confirms that the limitations of pure P2P approaches when faced with symmetric NAT can be overcome through fallback mechanisms. The average 30% increase in latency measured in relay mode, however, indicates that centralized fallback comes at a cost; thus, the architecture’s prioritization of P2P is design-wise justified. Cipher’s architectural decisions reflect deliberate trade-offs within the well-known trilemma of security/privacy, usability, and degree of decentralization. The use of a centralized service like Firebase for peer discovery and signaling is a compromise on the ideal of pure decentralization; however, this choice significantly improves usability on mobile devices by solving one of the

most challenging problems of P2P: reliable peer discovery. Given that poor usability is considered the primary reason technically robust tools like PGP have historically failed to gain widespread adoption [20], [21], integrating a high-performance Go core with a modern Flutter interface is a deliberate strategy to overcome this barrier.

The most significant limitation of this study lies in the security layer. Although the deterministic derivation of the session key simplifies the implementation, it has not yet achieved the level of forward secrecy and post-compromise security provided by the Signal Protocol’s X3DH and Double Ratchet algorithms [11]. Similarly, compared to fully decentralized mobile P2P systems such as Tribler [15], Cipher relies on Firebase for discovery and relaying. These aspects necessitate positioning the system as a proof-of-concept and a performance case study; the contribution is a comprehensive engineering synthesis of proven technologies rather than a new cryptographic principle. RSA-4096 was selected for implementation simplicity and interoperability; future versions will evaluate Curve25519/X25519-based key exchange [22] to improve computational efficiency and forward secrecy.

When positioned among existing secure messaging systems, Cipher occupies a distinct point in the design space summarized in Section 2 and Table 3 [1]. The qualitative comparison requested by the review process is therefore provided; a rigorous same-laboratory quantitative comparison of latency, energy, and throughput with WhatsApp, Signal, Tox, and related systems remains an avenue for future work.

VI. CONCLUSION

In this study, Cipher—an end-to-end encrypted, hybrid peer-to-peer mobile instant messaging system—was designed, implemented, and evaluated as an alternative to the privacy, censorship, and single point of failure risks associated with reliance on a central server. The high-performance concurrency capabilities of the Go language and Flutter’s cross-platform UI power have been combined under a single umbrella via the Gomobile bridge; an E2EE layer based on RSA-4096, HMAC-SHA256, and AES-256-GCM, a composite transport layer using WebRTC/UDP/relay, and ACK-based presence mechanisms have been successfully integrated.

Field tests have shown that the composite transport layer increases connection success rates to over 95%, that the application layer’s fragmentation and rate-limiting algorithms ensure reliable file transfer over UDP, and that asynchronous initialization reduces startup time to under 1 second. The system’s main limitations include the inability to provide forward secrecy at the Signal level due to the deterministic derivation of the session key, the reliance on Firebase for peer discovery and relay, and the requirement for external configuration of the TURN server. Future work plans to conduct expanded performance measurements incorporating X3DH-based key agreement, Double Ratchet session updates, DHT-supported

decentralized discovery, and desktop client integration. In conclusion, Cipher demonstrates that secure, mobile-first peer-to-peer messaging can be implemented with high usability while maintaining strong privacy guarantees.

REFERENCES

- [1]. C. Johansen, A. Mujaj, H. Arshad, and J. Noll, The Snowden Phone: A Comparative survey of Secure instant messaging mobile applications, *Security and Communication Networks*, 2021, 1-30.
- [2]. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, FreeNet: a distributed anonymous information storage and retrieval system, *Lecture Notes in Computer Science*, 2001, 46-66.
- [3]. N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, SoK: Secure messaging, *Proc. IEEE Symposium on Security and Privacy*, 2015, 232-249.
- [4]. P. Kumar, R. Narayan, and E. Rasid, Android Forensics: A Literature Review of Methodologies and Tool Efficacy, *International Journal of Advanced Networking and Applications*, 17, 2026, 7045-7054.
- [5]. E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, A survey and comparison of peer-to-peer overlay network schemes, *IEEE Communications Surveys & Tutorials*, 7, 2005, 72-93.
- [6]. P. Maymounkov and D. Mazières, Kademlia: a Peer-to-Peer information system based on the XOR metric, *Proc. International Workshop on Peer-to-Peer Systems*, 2002, 53-65.
- [7]. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, *ACM SIGCOMM Computer Communication Review*, 31, 2001, 149-160.
- [8]. A. Rowstron and P. Druschel, Pastry: scalable, decentralized object location, and routing for Large-Scale Peer-to-Peer systems, *Proc. IFIP/ACM International Conference on Distributed Systems Platforms*, 2001, 329-350.
- [9]. S. Ratnasamy, P. Francis, P. Handley, R. Karp, and S. Shenker, A scalable content-addressable network, *ACM SIGCOMM Computer Communication Review*, 31, 2001, 161-172.
- [10]. W. Diffie and M. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, 22, 1976, 644-654.
- [11]. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, A formal security analysis of the signal messaging protocol, *Journal of Cryptology*, 33, 2020, 1914-1983.
- [12]. T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, How secure is TextSecure?, *Proc. IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, 457-472.
- [13]. B. Ford, P. Srisuresh, and D. Kegel, Peer-to-peer communication across network address translators, *Proc. USENIX Annual Technical Conference*, 2005, 179-192.
- [14]. K. Ermoshina, F. Musiani, and H. Halpin, End-to-end encrypted messaging protocols: An overview, *Proc. International Conference on Internet Science (INSCI)*, 2016, 244-254.
- [15]. J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips, TRIBLER: a social-based peer-to-peer system, *Concurrency and Computation Practice and Experience*, 20, 2007, 127-138.
- [16]. B. A. Buhari, H. Abdulkadir, S. A. Ahmad, R. Sulaiman, M. M. Umar, S. Shehu, A. Almu, A. M. Abubakar, and J. O. Nwoji, Performance and Security Analysis of Symmetric Data Encryption Algorithms: AES, 3DES and Blowfish, *International Journal of Advanced Networking and Applications*, 16, 2025, 6473-6486.
- [17]. R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, 21, 1978, 120-126.
- [18]. D. A. McGrew and J. Viega, The security and performance of the Galois/Counter Mode (GCM) of operation, *Proc. INDOCRYPT*, 2004, 343-355.
- [19]. Y. Wang, Z. Lu, and J. Gu, Research on symmetric NAT traversal in P2P applications, *Proc. International Multi-Conference on Computing in the Global Information Technology (ICCGI)*, 2006, 59.
- [20]. A. Whitten and J. D. Tygar, Why Johnny can't encrypt: A usability evaluation of PGP 5.0, *Proc. 8th USENIX Security Symposium*, 1999.
- [21]. R. Abu-Salma, M. A. Sasse, J. Bonneau, A. Danilova, A. Naiakshina, and M. Smith, Obstacles to the adoption of secure communication tools, *Proc. IEEE Symposium on Security and Privacy*, 2017, 137-153.
- [22]. D. J. Bernstein, Curve25519: New Diffie-Hellman speed records, *Proc. International Workshop on Public Key Cryptography (PKC)*, 2006, 207-228.

BIOGRAPHIES

Miraç Deprem received his vocational high-school diploma in information technologies and database programming from Halit Narin Vocational and Technical Anatolian High School, Istanbul, in 2021. He is currently pursuing a B.Sc. degree in Computer Engineering at the Faculty of Technology, Isparta University of Applied Sciences, Türkiye (2022–2026). His internships include information-technology work at Çetin Group (2025–2026) and software development at Aksiyon Bilişim (2025). His research interests include peer-to-peer networks, end-to-end encryption, and mobile systems. He is a co-developer of the Cipher messaging prototype described in this paper.

Eren Ali Koca is currently pursuing a B.Sc. degree in Computer Engineering at the Faculty of Technology, Isparta University of Applied Sciences, Türkiye (2022–2026). He is a software developer at TABY, working on banking and financial web applications with React and .NET Core REST APIs, and has internship and freelance experience in React Native and native iOS (UIKit/SwiftUI, MVVM). He completed the Turkcell Geleceği Yazarlar Swift Bootcamp and has published iOS applications on the App Store. His interests include mobile systems and software architecture. He is a co-developer of the Cipher messaging prototype described in this paper.

Ahmet Ali Süzen received the B.Sc. degree in Electronics and Computer Education (2011) and in Computer Engineering (2021), the M.Sc. degree in Electronics and Computer Education (2013), and the Ph.D. degree in Computer Engineering (2018) from Süleyman Demirel University, Türkiye. He is an Associate Professor in the Department of Computer Engineering, Faculty of Technology, Isparta University of Applied Sciences, where he also serves as Head of Department. His research interests include cybersecurity, cryptography, artificial intelligence, and computer networks. He has authored books and numerous journal articles in these areas.