http://www.moderntechno.de/index.php/meit/article/view/meit42-02-056
DOI: 10.30890/2567-5273.2025-42-02-056

UDC 004.05:004.738

DEVELOPMENT OF A PROTOTYPE APPLICATION FOR EXCHANGING ENCRYPTED MESSAGES ON TELEGRAM WITH THE CHOICE OF THE OPTIMAL ENCRYPTION ALGORITHM

Oleksandra Basarab
master's student
ORCID: 0009-0000-5146-775X
Maria Curie-Skłodowska University
Oleksandr Basarab
c.t.s., as.prof
ORCID: 0000-0002-2852-9534
The Bohdan Khmelnytskyi National Academy of the State Border Guard Service of Ukraine

Abstract. *This paper presents a prototype application that provides application-layer encryption for messages delivered through Telegram. The workflow is intentionally simple: a sender encrypts a message locally, transmits it as an encrypted file payload via Telegram, and the receiver downloads and decrypts it inside the application. The prototype supports a configurable set of symmetric and asymmetric algorithms and derives session keys via Diffie-Hellman key agreement followed by key derivation. To understand practical performance, the study combines two evaluation stages: controlled, algorithm-only benchmarking and end-to-end measurements over the real transport path under both unconstrained and bandwidth-limited conditions. The results show that user-perceived delay is primarily influenced by transport and payload size, while the cryptographic cost of symmetric algorithms is comparatively small in the full workflow. Asymmetric encryption applied directly to message content introduces additional overhead and is less suitable for bandwidth-constrained usage. The paper concludes with practical guidance for selecting encryption mechanisms in Telegram-based transport scenarios and outlines straightforward security improvements for future iterations.*

Key words: *Diffie-Hellman key agreement, encryption algorithms, encrypted file transfer, public-key cryptography, symmetric cryptography.*

Introduction.

Messaging platforms are widely used because they are convenient, accessible, and reliable as delivery channels. In many practical situations, however, users and developers still seek an additional protection layer that remains independent of the platform's internal security mechanisms and user settings. This expectation is common in scenarios where sensitive information is exchanged over a channel that is not fully controlled by the communicating parties.

One pragmatic approach is application-layer encryption: the message is protected before it is handed to the messaging service and stays protected while it is transported. In this model, the platform is treated primarily as a carrier, while confidentiality is

ensured by encryption performed outside the platform. The main question is not only whether such encryption can be added, but also how it affects usability. For user-facing communication tools, responsiveness matters: delays influence perceived quality and can discourage adoption. This becomes especially relevant under limited bandwidth, where a transport path may dominate the overall message delay.

Several studies and standards-based discussions describe contemporary encryption choices and compare cryptographic primitives, often concluding that modern symmetric encryption is the practical default for bulk data, while legacy designs serve mainly as contrast baselines and public-key schemes are expensive when applied directly to message payloads [1-3]. These works are valuable for understanding algorithm-level properties and performance. However, many of them evaluate ciphers in isolation (e.g., micro-benchmarks or file-level tests) and do not consider a complete application workflow that includes a real transport path and constrained bandwidth. As a result, they only partially answer the question that matters in practice: what the user actually experiences when encrypted content is sent through a messaging platform.

This paper addresses this problem by proposing and evaluating a prototype application for encrypted message exchange over Telegram. The system encrypts a message locally, sends it as an encrypted file payload through Telegram, and decrypts it on the receiver side. It allows selecting between multiple encryption algorithms to compare their practical impact within the same end-to-end workflow. Session keys are established using Diffie-Hellman key agreement and a key derivation step, which reflects common practices in secure communication systems.

The study follows a two-stage methodology. First, algorithm-only benchmarking isolates the computational overhead of encryption and decryption. Second, end-to-end experiments measure user-visible latency in realistic conditions, including bandwidth constraints. This combination makes it possible to distinguish cryptographic overhead from transport-dominated delay and to formulate practical guidance for cipher selection in Telegram-based delivery scenarios.

To achieve this goal, the following sub-tasks were completed:

- review contemporary encryption protocols;

- justify and choose evaluation criteria for comparing encryption protocols;
- develop a research methodology for the selected protocols according to the chosen criteria;
- implement a benchmarking program for controlled, algorithm-only measurements;
- implement a messaging application that exchanges encrypted messages with a configurable choice of algorithms and DH-based session keys over Telegram;
- experimentally evaluate the effectiveness and performance of the developed application under constrained and unconstrained bandwidths.

Main text.

Selection of encryption algorithms and evaluation criteria. The prototype supports a configurable set of encryption algorithms to reflect both a practical baseline and meaningful comparison cases. AES is treated as the main modern symmetric option for message payload protection [4], while DES, 3DES, Blowfish, and TEA are included as legacy or alternative references [5-8]. RSA is included as an asymmetric reference to illustrate the typical cost of applying public-key encryption directly to message content rather than to small key material [1-3,9].

The evaluation criteria were defined to capture both algorithm behaviour in controlled conditions and the same algorithms observed inside an end-to-end exchange over Telegram. In the controlled benchmark stage, each algorithm was compared using:

- encryption time and decryption time measured as wall-clock time for repeated processing of short messages;
- CPU time and overall CPU utilisation to characterise computational load;
- memory usage captured as peak resident set size (RSS) above a baseline;
- ciphertext size/expansion, including required metadata such as IV and padding, and block structure in the asymmetric case.

Short, chat-like plaintext messages of different sizes were used to represent typical messaging inputs. Repeated batch processing was applied to make differences measurable and comparable across algorithms.

Session key establishment and message protection model. The prototype uses a session-key model consistent with common secure-communication practice. A shared secret is established with Diffie-Hellman key agreement and then expanded into application keys using a key derivation step. Session keys are created per conversation and reused within that conversation.

For message protection, symmetric ciphers follow a consistent «encrypt → send → decrypt» flow, with fresh IVs generated from a cryptographically secure random source where applicable. RSA is treated as a special case because message content must be processed in chunks due to size limits of public-key encryption; this also affects ciphertext structure and payload size.

Application workflow and implementation overview. The application implements a repeatable workflow: a sender writes a message, the application encrypts it locally, packages the ciphertext as a binary «enc» payload, sends it through Telegram, and the receiver downloads and decrypts it inside the application.

The system follows a web-based client–server structure. The backend is responsible for cryptography, Telegram I/O, and data collection for analysis. Encrypted payloads are exchanged as files, and temporary «enc» files are created during send/receive and removed afterwards.

To support evaluation, the system records sizes and timestamps for each message so that local cryptographic overhead and transport-dominated delay can be analysed separately. Measurements are stored per message and exported for offline processing.

Evaluation methodology. The evaluation was performed in three connected steps.

First, a controlled benchmark was used to isolate cryptographic routines from transport effects [10]. For each algorithm, encryption and decryption were executed repeatedly on short messages in batch form. The benchmark recorded encryption/decryption wall time, CPU time and CPU utilisation, peak memory (RSS), and ciphertext size/expansion. Runs were repeated and aggregated to reduce noise and to avoid conclusions based on single, unstable measurements.

Second, the selected algorithms and the session-key mechanism were integrated into the application. A user can select the encryption option at conversation creation,

and the chosen algorithm is stored and applied consistently for subsequent messages in that conversation. Encrypted messages are packaged as ".enc" files, with a clear layout that includes the parameters needed for decryption on the receiver side.

Third, the integrated system was evaluated in an end-to-end exchange path. Measurements captured the complete workflow: local encryption, upload/send through Telegram, download/fetch by the receiver, and local decryption. Network conditions were shaped using three bandwidth profiles, allowing the same workflow to be compared under unconstrained and constrained transport. For each message, the system recorded plaintext/ciphertext sizes, local crypto times, and end-to-end timestamps; the resulting dataset was exported to CSV for offline analysis.

Because file transfer through Telegram introduces variability at the single-message level, transport behaviour was analysed separately from local crypto time. The interpretation emphasised conservative user experience by focusing on upper-tail delay patterns rather than relying only on averages, and by relating transport delay to bandwidth profile and payload size before composing total end-to-end time.

Descriptive results and interpretation. The controlled benchmark indicates that symmetric ciphers have relatively small per-message computational overhead on short inputs, while RSA is substantially heavier, especially on decryption, and produces larger payloads due to chunked processing.

In the end-to-end Telegram workflow, the overall delay is dominated by the transport path, and this dominance becomes stronger as bandwidth is constrained. In this setting, the choice among symmetric ciphers has limited influence on what the user feels for a single message, while RSA is noticeably less suitable for encrypting message content directly because it combines higher computation cost with larger payload size. In practical terms, RSA tends to increase worst-case end-to-end delay by a small but consistent margin, which becomes more visible when many messages are exchanged.

These observations support a common design direction: use symmetric encryption for message payloads and treat public-key mechanisms primarily as tools for key establishment or key protection.

Summary and conclusions.

This paper proposed and evaluated an application-layer encryption approach for sending messages through Telegram while keeping message content protected outside the transport platform. A prototype application was implemented with a clear workflow: local encryption, delivery as an encrypted file payload through Telegram, and local decryption on the receiver side. The system supports multiple encryption algorithms and derives session keys via Diffie-Hellman key agreement followed by key derivation, enabling consistent comparison within the same end-to-end workflow.

A two-stage evaluation methodology was applied. Controlled algorithm-only benchmarking characterised cryptographic overhead, while end-to-end experiments through Telegram measured user-visible latency under both unconstrained and bandwidth-limited conditions. The main conclusion is that transport effects and payload size typically dominate perceived delay in bandwidth-constrained messaging. As a result, differences among symmetric algorithms have limited practical influence on user experience in this workflow, while asymmetric encryption applied directly to message content introduces additional overhead and is less suitable for constrained transport. The findings therefore favour a session-based design that relies on symmetric encryption for payload protection and uses public-key methods for key establishment or key protection.

The work also indicates clear directions for improvement. Future iterations can strengthen security completeness by adding authenticated encryption with associated data (AEAD), for example AES-GCM [11], to provide integrity and tamper detection together with confidentiality. Further enhancements can also extend session handling and message-processing safeguards without changing the overall workflow or the evaluation framework described in this paper.

References:

1. Ebrahim, M., Khan, S. and Khan, U. (2014) Symmetric Algorithm Survey: A Comparative Analysis, *arXiv preprint*, arXiv:1405.0398.

2. Muhammed, N. et al. (2017) Evaluation of Four Encryption Algorithms for

Viability, Reliability, and Performance Estimation, *Proceedings* (online preprint).

3. Girish, N., Akshay, B. and Prathamesh, R. (2020) Comparative Analysis of Encryption Algorithms Against Text Files, Conference/Workshop paper or journal article (publication details not provided).

4. FIPS 197; Advanced Encryption Standard (AES); National Institute of Standards and Technology; U.S. Department of Commerce: Gaithersburg, MD (26 November 2001).

5. FIPS 46-3; Data Encryption Standard (DES) (Withdrawn); National Institute of Standards and Technology; U.S. Department of Commerce: Gaithersburg, MD (25 October 1999; withdrawn 19 May 2005).

6. NIST SP 800-67 Rev. 2; Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher; National Institute of Standards and Technology; U.S. Department of Commerce: Gaithersburg, MD (November 2017).

7. Schneier, B. (1994) Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), in *Fast Software Encryption (FSE '93). Lecture Notes in Computer Science*, Vol. 809. Springer.

8. Wheeler, D.J. and Needham, R.M. (1995) TEA: A Tiny Encryption Algorithm, in *Fast Software Encryption (FSE '94). Lecture Notes in Computer Science*, Vol. 1008. Springer.

9. Rivest, R.L., Shamir, A. and Adleman, L. (1978) A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, 21(2), pp. 120–126.

10. Jorge, R. (2020) cryptographic-benchmarking-aes-rsa-sha256. GitHub repository. (Accessed: 21 June 2024).

11. NIST SP 800-38D; Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC; National Institute of Standards and Technology; U.S. Department of Commerce: Gaithersburg, MD (28 November 2007).

Article sent: 20.12.2025