LINE-Break: Cryptanalysis and Reverse Engineering of Letter Sealing

Anonymous Author(s)

Abstract

We present a security analysis of the messaging service known as LINE, a popular messaging platform used daily by millions of users in Southeast Asia – most notably Japan, Taiwan, Thailand, and Indonesia. More specifically, we focus on its underlying custom end-to-end encryption (E2EE) protocol, known as Letter Sealing v2. Our findings show that Letter Sealing allows a TLS Man-in-the-Middle attacker or malicious server to violate integrity, authenticity, and confidentiality of communications. The stateless design of the protocol allows message replay, reordering, and blocking attacks without the user being notified. The lack of origin authentication facilitates impersonation attacks, in which the authorship of messages in one-to-one or group chats can be forged by malicious users colluding with the adversary. Lastly, stickers and URL previews present a notable leakage of plaintext, which leads to a violation of confidentiality. To verify the correctness of our findings, we mounted a Man-in-the-Middle attack on an iOS device, yielding the device's outgoing traffic and the corresponding server responses. Utilizing this setup, we experimentally verified our attacks against the authentic LINE application and an independent implementation. We discuss our findings in comparison to the state-of-the-art E2EE protocols, and conclude that Letter Sealing does not satisfy the requirements expected from a modern E2EE messaging protocol.

1 Introduction

End-to-end encrypted (E2EE) instant messaging is one of the most widely deployed cryptographic protocols in history, with millions of global users exchanging billions of encrypted messages every day. Much of this success is attributed to the Double-Ratchet algorithm [15], a core part of applications such as Signal, WhatsApp, Wire and Facebook Messenger. It combines a public-key ratchet based on the Diffie–Hellman key exchange with a symmetric-key ratchet based on a key derivation function. The resulting combination enables advanced security properties, such as forward secrecy and some form of post-compromise security (self-healing) [4]. Despite the security guarantees given by state-of-the-art E2EE protocols, some popular messengers have chosen to deploy customized protocols with unclear security guarantees and mixed results, such as Telegram [1, 2, 12], Threema [14] and LINE.

LINE is a free instant messaging service launched in Japan in 2011. It was later deemed a Super App due to the large variety of services it provides, such as a digital wallet, news, video-on-demand, and even games. LINE is widely used, with an estimated one billion annual users by 2025 and approximately 200 million active monthly users [16]. The application is used most extensively in Southeast Asia, covering roughly 80% to 85% of the population of Japan from 2017 to 2024 [10, 17], and covering 99.5% of people in their 20s in Japan [9, 11]. Due to the widespread adoption, it is used for e-government services within Japan [5], and its revenue peaked at $2.36 billion in 2021 [16]. Previously owned by LINE Corporation, it is now under the ownership of LY Corporation following a merger with Yahoo! Japan and two other companies. The messaging application incorporates a custom end-to-end encryption (E2EE) protocol, coined Letter Sealing (LS), originally developed throughout 2014-2016 and launched at scale in 2016 [6, 13].

Previous work has documented several weaknesses in the original design of the protocol, named Letter Sealing v1 (LSv1), featuring attacks such as replay, forgery, and impersonation attacks [10]. The protocol was updated to v2 in 2019 and has since been updated again in 2021 [6]. With the design of LSv2, LY Corporation claims to have *all the issues resolved* [6, 10], which stands in stark contrast to the findings in this work.

Contributions. We continue the effort of analyzing "cryptography in the wild" [3] and present a security analysis of LSv2, from an analysis of the white paper [6], interactions with the live LINE protocol and reverse engineering of its implementation. The experiments were conducted using a Man-in-the-Middle (MitM) setup against an iOS client using a rogue root certificate, such that the adversary would be capable of tampering with protocol metadata in the same way as a malicious server. We interact with LINE servers through both the official application and an independent JavaScript implementation of the client-side portion of the protocol, called LINEJS[1]. We present the following attacks:

- *Equivocation attacks*: messages can be replayed, reordered, or dropped without the endpoints being aware, violating transcript consistency. This is inherent to the stateless nature of the protocol, which is insufficiently mitigated by easy-to-bypass server-side countermeasures. We show further attacks against integrity by tampering with read receipts.
- *Impersonation attacks*: the authorship of messages in one-to-one and group chats can be forged by a malicious user colluding with the adversary. This is a direct consequence of the way (group) keys are generated and managed in LSv2, and the lack of origin authentication measures. Combined with the previous attacks, we show how the adversary would theoretically be able to forge communications among a subset of parties in a group chat, or infiltrate a group chat and manipulate its following communication.
- *Plaintext leakage attacks*: there is substantial leakage of plaintext through usability features, such as stickers and URL previews. While LSv2 documentation notifies the user of some of the privacy-usability trade-offs around these features, we find that in case of an E2E Adversary, the leakage goes beyond what one would expect from the public documentation.

[1] LINEJS library in Javascript to create bots: `https://github.com/evex-dev/linejs`

Our findings were disclosed in June to the LY Corporation Computer Security Incident Response Team and later confirmed by the LSv2 Team, with a statement submitted as supplementary material.

2 Background

Letter Sealing is claimed to provide E2EE for text messages and media streams, ensuring that "*no third parties or LINE Corporation can decrypt private calls and messages*" [6]. The company asserts confidentiality, partial forward security (between clients and servers only through TLS), integrity and authenticity. LSv2 employs standardized primitives, such as ECDH for static key exchange using X25519, and AES256-GCM for payload encryption.

Protocol overview. When launching a fresh LINE client and authenticating with the LINE server, the client generates an ECDH key pair consisting of a secret key and a public key (sk, pk). The secret key is stored in the application's private storage, whereas the public key is registered on the LINE server under a unique key identifier. Users can verify public keys out of band using the public key fingerprints displayed within the app [6].

In a one-to-one chat between users denoted as U_i and U_j, with their corresponding keys (pk_{U_i}, sk_{U_i}), and (pk_{U_j}, sk_{U_j}), a shared secret is generated as $\text{ECDH}(sk_{U_i}, pk_{U_j}) = \text{ECDH}(sk_{U_j}, pk_{U_i})$, which is then concatenated with a salt to derive the encryption key through SHA256. When sending a message, a 12-byte nonce is constructed as the concatenation of an 8-byte per-chat counter (tracking the sender's number of messages sent) and a 4-byte random value. The sender forms additional authenticated data (AAD) containing metadata such as the sender and recipient identifiers, their key identifiers, protocol version, and content type. The plaintext message is then encrypted using AES-GCM, producing ciphertext and an authentication tag. The server routes the ciphertext to the recipient, who reconstructs the session key using their private key and the sender's public key, then verifies the tag and decrypts the message.

Group chats extend the one-to-one protocol through a shared *group key* distributed via pairwise ECDH exchanges. When a user U_i sends the first message to a group, they generate a fresh ECDH key pair (sk_g, pk_g) and designate sk_g as the *group secret key*. The initiator retrieves public keys $\{pk_{U_j}\}$ of all members from the server and derives pairwise keys K_j derived from $\text{ECDH}(sk_{U_i}, pk_{U_j})$ using a salt and nonce (similarly to one-to-one chats), then encrypts sk_g for each member using AES256-CBC, producing $\{E_{K_j}(sk_g)\}$, which are uploaded to the server for distribution. Each recipient decrypts their entry to recover sk_g and join the shared group context. Subsequent group messages derive per-message encryption keys as $\text{ECDH}(sk_g, pk_{U_i})$, combined with a random salt to form the AES-256-GCM key. Nonces and AADs follow the one-to-one structure, except that recipient identifiers are replaced by the group identifier and its key identifier. Any member holding sk_g can decrypt messages from any other member. Changes in group membership trigger a key rotation: a new (sk_g, pk_g) is generated and redistributed pairwise.

Threat Model. Our security analysis assumes several adversarial settings capturing both external and internal threats to an E2EE messenger. These models reflect realistic attack surfaces for LSv2:

- *Malicious Server (MS):* The strongest adversary, controlling the LINE servers and capable of intercepting, modifying, replaying, or injecting any ciphertext exchanged between clients. This captures the case of a compromised or malicious service provider and is the primary threat model for E2EE, as the protocol should preserve confidentiality and authenticity even against an insider.
- *MitM Adversary:* A network attacker positioned between a client and the server, able to observe and manipulate traffic but without full server control. The MitM model is relevant when transport-layer protections (e.g., TLS) are disabled, bypassed, or compromised. Furthermore, an MitM attacker can emulate a malicious server by manipulating metadata.
- *Malicious User:* A legitimate participant in a one-to-one or group chat who misuses their possession of shared (group) keys to manipulate protocol state and break another user's security properties (e.g., forging or replaying messages).

3 Findings

We split the discussion by the affected security property.

Integrity. LSv2 lacks a cryptographically-bound protocol state. Each client maintains an independent message counter that is neither synchronized nor validated by the endpoints. Nonces and counters are thus accepted without verification, and server-side modifications further obscure state tracking.

Our experiments reveal that LSv2 messages can be replayed, reordered, or blocked by adversaries under both the MS and MitM threat models, violating transcript consistency. Because message counters are unauthenticated metadata, an attacker can craft valid ciphertexts with altered counters to bypass server-side mitigations. Certain counter encodings, resulting from the protocol serialization format, allow unlimited replays. Consequently, a compromised or malicious server can indefinitely resend prior ciphertexts, and a MitM adversary can inject replayed messages indistinguishable from legitimate traffic. An adversary can also block or reorder messages, since the recipient cannot detect message loss, and is unable to verify correct ordering.

We have simulated these attacks by mounting a man-in-the-middle attack between the authentic app and the LINE server under an MitM setting. As such, we have been able to carry these out using only a subset of the power a malicious server would be have. We have not modified anything else beyond giving ourselves the ability to modify or repeat the ciphertexts in transit.

We present evidence of the practicality of the attacks in video:

- *Replay*[2]: The adversary records the ciphertext being sent from a user to another, and is able to simply repeatedly re-send (a slightly modified) copy multiple times.
- *Reordering*[3]: The adversary can reorder messages (while changing timestamps) or block them indefinitely, making it possible to re-arrange messages in transit at will.

When a party receives a message and opens the chat to read it, their client sends a *read receipt* to the sender. As LS is disabled for read receipts, they can be forged by the adversary. This issue

[2]Replay attack: `https://youtu.be/gjMLL3oAt0w`
[3]Reordering attack: `https://youtu.be/hzoRdv8FIi8`

further powers the message blocking and impersonation attacks. The same vulnerabilities are present in group messaging, mirroring the weaknesses of the one-to-one design.

Authenticity. We demonstrate impersonation attacks in both one-to-one and group settings. The absence of any origin authentication, such as digital signatures or key-confirmation steps, allows an adversary with valid session keys to fabricate ciphertexts that recipients will decrypt as genuine messages from another user.

In one-to-one chats, this is feasible for malicious users in collusion with MS or MitM adversaries. Due to the use of symmetric key material for encryption, the malicious user can prepare a message such that it is cryptographically valid and appears to originate from another party, and the malicious server must reorder the positioning of metadata fields in transit. We note that this falls into the *deniability* property of E2EE messaging, but there is a subtle difference when comparing to other protocols. In the Signal protocol, a user can always append a message into the symmetric ratchet to make it appear as if received from another user. LSv2 allows a user colluding with the adversary to forge an incoming message at any part of the transcript, or to forge a message coming from another user and collude with the adversary to deliver it to their device as if creating evidence of authorship, because keys are not bound to the direction of communication.

In group chats, the issue is more concerning because any legitimate group member can compute the key material necessary to impersonate any other group member, a direct consequence of the *shared* group keys. The symmetric keys used for encryption and authentication do not protect against manipulating the sender and recipient fields within a message, crafting a cryptographically valid message from another party. The practicality of the attack boils down to routing, server-side and client-side mitigations.

Our experimental setup shows the attack feasibility from the indistinguishability between forged and legitimate traffic. We demonstrate the attack in a one-to-one chat in a sequence of videos. The first two videos show an honest transmission from Alice [4] and correct decryption by Bob [5]. The last three videos show Alice manufacturing an evil message impersonating Bob's authorship [6] that is delivered to Bob and is correctly decrypted after the adversary adjusts the metadata accordingly [7][8].

Confidentiality. The lack of protection for stickers allows the adversary to capture sticker lookups, leaking information about content in encrypted chats. The leakage is threefold:

(i) upon writing a matching keyword, some stickers are previewed automatically. If this is the first preview, the sticker is downloaded[9].
(ii) upon previewing a sticker, the animation for the sticker is downloaded, along with two related stickers.
(iii) upon sending a sticker, the full package is downloaded[10].

Sending a URL through LINE leaks the URL to the server in order to create previews [11]. The URL will similarly be fetched by all members of the chat. This video demonstrates how the URL entered by the user is trivially readable by a MitM adversary as it is sent to the server. This also clearly shows that the URL is leaked to the server, despite being sent as a message in an LSv2 chat.

LY Corporation claims in [7, 8] that the sticker lookup feature is *anonymous*. While it is true that the user's authentication token is not present in the lookup, the user-agent and language fields form an identifier that, although not necessarily unique, should aid in tracking specific devices. Specifically, LINE formats user-agent headers with three partial entries, e.g., `LINE/2025.415.1834 CFNetwork/3826.400.120 Darwin/24.3.0`. These fields represent the application version, the networking framework version, and the operating system version, respectively. The language field contains the preferred languages and a preference score. We believe these values constitute enough of an identifier to substantially challenge the claim of anonymity.

Putting all together. Under the threat model of a malicious group member colluding with the adversary, it is feasible to craft full conversations stealthily. A malicious server can manipulate metadata at will, drop messages for certain users, and forge read receipts. The malicious group member (or the server) can add a metadata status field which disables recipient notifications for that message. From these, one can build a large-scale, high-impact impersonation attack that crafts full conversations for any subset of group members. Assuming that the group stays consistent throughout the attack, all members can read the full conversation, which is indistinguishable from a legitimate one. The adversary fully controls the contents of each message and can specify who will be displayed as the sender of each message. Finally, the malicious server can withhold messages from certain victims, creating inconsistent views of the conversation or leaving the impersonated victims unaware of the attack altogether.

Because group key rotations are handled server-side without cryptographic binding to group membership, we conjecture that malicious users can trigger illegitimate key rotations to cause denial-of-service, maintain access after removal, or even gain illegitimate group membership. Finally, the reuse of group keys across communication types (e.g., VoIP) exposes a potential avenue for passive eavesdropping on group calls.

4 Conclusion

Overall, our findings demonstrate that LSv2 fails to provide several core E2EE properties under realistic adversarial models. In particular, there is no end-to-end forward secrecy. While AES-GCM and Curve25519 are sound primitives, their composition in a stateless, unauthenticated protocol undermines the intended security. Mitigations should include explicit message authentication codes over protocol metadata, cryptographically enforced counters, per-message key evolution (ratcheting), and authenticated key-confirmation. Adoption of a modern construction such as the Signal Double-Ratchet or Sender Keys would resolve most identified issues.

[4] Impersonation attack 1/5: `https://youtu.be/vaXYXNK9hEc`
[5] Impersonation attack 2/5: `https://youtu.be/RqDh81Mqmwk`
[6] Impersonation attack 3/5: `https://youtu.be/RfsXFQDvl3k`
[7] Impersonation attack 4/5: `https://youtu.be/Pv5FvuBuC98`
[8] Impersonation attack 5/5: `https://youtu.be/hWN-iYc5coQ`
[9] Sticker leakage 1/2: `https://youtu.be/yBHANhla-ms`
[10] Sticker leakage 2/2: `https://youtu.be/QpSqcJ9YueI`
[11] URL leakage: `https://youtu.be/P_a8rR9AP-U`

References

[1] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, Eyal Ronen, and Igors Stepanovs. 2025. Analysis of the Telegram Key Exchange. In *EUROCRYPT (8) (Lecture Notes in Computer Science, Vol. 15608)*. Springer, 212–241.

[2] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. 2023. Four Attacks and a Proof for Telegram. *IACR Cryptol. ePrint Arch.* (2023), 469.

[3] Martin R. Albrecht and Kenneth G. Paterson. 2024. Analyzing Cryptography in the Wild: A Retrospective. *IEEE Secur. Priv.* 22, 6 (2024), 12–18.

[4] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A Formal Security Analysis of the Signal Messaging Protocol. *J. Cryptol.* 33, 4 (2020), 1914–1983.

[5] LY Corporation. 2020. Smart city transformation of local governments (in Japanese). https://linegov.com/service/partner.php.

[6] LINE Corporation. 2021. LINE Encryption Overview. https://scdn.line-apps.com/stf/linecorp/en/csr/line-encryption-whitepaper-ver2.1.pdf.

[7] LINE Corporation. 2022. LINE Encryption Report. https://linecorp.com/en/security/encryption/2022h1.

[8] LY Corporation. 2024. LINE Encryption Report. https://www.lycorp.co.jp/en/privacy-security/security/transparency/encryption-report/2024/.

[9] Statista Research Department. 2025. Share of people who use LINE in Japan in fiscal year 2024, by age group. https://www.statista.com/statistics/1077541/japan-line-penetration-rate-by-age-group/.

[10] Antonio M. Espinoza, William J. Tolley, Jedidiah R. Crandall, Masashi Crete-Nishihata, and Andrew Hilts. 2017. Alice and Bob, who the FOCI are they?: Analysis of end-to-end encryption in the LINE messaging application. In *FOCI @ USENIX Security Symposium*. USENIX Association.

[11] SB Telecom Europe. 2024. 2024 Social Media in Japan. https://www.digitalmarketingforasia.com/wp-content/uploads/2024/07/2024-Social-Media-in-Japan.pdf.

[12] Jakob Jakobsen and Claudio Orlandi. 2016. On the CCA (in)Security of MTProto. In *SPSM@CCS*. ACM, 113–116.

[13] LINE. 2016. Letter Sealing gets enhanced! https://line-en-official.weblog.to/archives/1060089042.html.

[14] Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong. 2023. Three Lessons From Threema: Analysis of a Secure Messenger. In *USENIX Security Symposium*. USENIX Association, 1289–1306.

[15] Trevor Perrin and Moxie Marlinspike. 2016. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/. Revision 1, 2016-11-20.

[16] Leonardo Pires. 2024. LINE Revenue and Growth Statistics. https://usesignhouse.com/blog/line-stats/.

[17] World Population Review. 2025. Line Users by Country 2025. https://worldpopulationreview.com/country-rankings/line-users-by-country.