Technical Whitepaper

LINE Encryption Overview

v2.2, December 2025

LY Corporation

Copyright

Revision history

Version	Date	Notes
2.2	2025-12-16	Update for Media E2EE
2.1	2021-11-29	Add updates
2.0	2019-10-28	Update for Letter Sealing v2
1.0	2016-09-29	Initial Publication

Contents

Introduction

Letter Sealing is the common name of all end-to-end encrypted (E2EE) protocols integrated in LINE's messaging and VoIP services. This whitepaper offers in-depth details to Letter Sealing, encryption protocols and algorithms used in LINE's messaging and VoIP platform.

The scope of this document extends to LINE clients for Android and iOS. Clients targeting other platforms may have different implementation. The protocols in this document are integrated in LINE v6.7 and later versions.

Letter Sealing v2 has been integrated in LINE from v8.15 for iOS mobile clients, v8.17 for Android mobile clients, and v2.6 for LINE Lite clients.

Targeting security engineers and developers, this whitepaper assumes that readers have a strong understanding of encryption technology.

Account Registration

Signing up for a LINE account requires users to provide a valid phone number[a] and register an account password. Users can add their email address after registration. The email address and password registered are used in account migration[b], logging in to LINE Desktop clients and for accessing LINE's web-based services.

If the user chooses to register an email address, LINE requests the user to verify the ownership of the email address by sending a randomly generated 4-digit verification code to the specified email address. The user then is required to enter this code into the LINE client or alternatively, click the verification link provided with the code, on their mobile device. If and only if verification is successful, the user account is enabled to use LINE authentication (namely, LINE Login) via email address and password.

Client-to-Server Transport Encryption

LINE mobile clients no longer use SPDY 2.0 as the main transport protocol, as SPDY 2.0 is deprecated. So is the earlier transport layer encryption implemented in LINE. The new main protocol is HTTP2[1].

LINE is in the process of migrating to TLS 1.2 and TLS 1.3[2]. See the transparency report[3] for the migration status.

[a] Creating a new LINE account with a Facebook account is obsolete from April 2020.
[b] You can transfer LINE accounts among devices. For more information, see https://guide.line.me/en/migration/

Letter Sealing Overview

Letter Sealing implies all end-to-end encrypted (E2EE) protocols integrated in LINE's messaging and VoIP services. Letter Sealing is applied to text messages, location messages, and 1:1 audio & video calls. The scope of application is subject to change. For up-to-date information on scope, see the Transparency Report[3] and the LINE Encryption Report[4].

LINE encrypts messages locally on a client device before sending messages to LINE's messaging server. Such encrypted messages can only be decrypted by the intended recipient. Letter Sealing is applied only to message payloads; message metadata such as sender ID and recipient ID is not encrypted.

Although the first version of Letter Sealing encountered message data integrity issues[5], Letter Sealing v2 guarantees stronger protection over messages, with all the issues resolved. Although not encrypted, the new version protects the integrity of message metadata.

Letter Sealing v2 is applied by default on 1:1 and group messages if the sending LINE client supports v2. If the receiving client does not support Letter Sealing v2, the sending client downgrades the protocol to Letter Sealing v1 and resends the message.

The main cryptographic algorithms used in Letter Sealing for messaging, supported data and metadata protection levels are as the following.

	Version 1	**Version 2**
Key exchange algorithm	ECDH over Curve25519[6]	
Message encryption algorithm	AES256-CBC	AES256-GCM[7]
Message hash function	SHA-256	N/A
Data authentication	AES-ECB with SHA-256 MAC	AES256-GCM
Message data	Encryption and integrity	
Message metadata	Not protected	Integrity

1:1 Message Encryption

To send encrypted messages, LINE clients go through a process of generating, registering, and exchanging keys and lastly encrypting messages. All units of this process are similar between the two versions of Letter Sealing. The difference lies in message encryption protocols.

Key Generation and Registration

Sending encrypted messages requires a Letter Sealing ECDH key pair. LINE clients generate a key pair if there is none for the LINE user on the device and save the pair securely in the application's private storage.

After generating a key pair, the LINE client registers the public key on a LINE's messaging server. The server then associates the key with the authenticated LINE user and returns a unique key ID to the client. This unique key ID is bound to the LINE user and represents the latest version of the user's public key.

A new key is generated and registered each time the LINE application is reinstalled or when the user migrates their account to a new device.

Client-to-Client Key Exchange

LINE clients participating in a chat must share a common cryptographic secret before exchanging encrypted messages. The process for obtaining a shared secret is as follows.

1. The sending client retrieves the current public key of the recipient.
2. The sending client passes its own private key and the recipient's public key to the ECDH algorithm to generate a shared secret.
3. The recipient generates the same shared secret using their own private key and the sender's public key.

The ECDH algorithm used in the process is as follows.

$$
\begin{aligned}
&SharedSecret \\
&= \mathrm{ECDH}_{\mathrm{curve25519}}(key_{private}^{user1}, key_{public}^{user2}) \\
&= \mathrm{ECDH}_{\mathrm{curve25519}}(key_{private}^{user2}, key_{public}^{user1})
\end{aligned}
$$

The whole process is transparent to users. There is no trusted third party who verifies public keys. Nevertheless, there is a way for users to verify the recipient. LINE enables users to view the fingerprint of their and the recipient's public key and thus verify the key out-of-band[a].

1:1 Message Encryption with Letter Sealing v1

LINE encrypts each message with a unique encryption key and IV (Initialization Vector).

Message sender

The sending LINE client encrypts the message in the following steps. The sending client:

1. Derives an encryption key and IV from the shared secret calculated and a randomly generated 8-byte salt:

[a] https://help.line.me/line/?contentId=20004441

$$salt = random_{secure}(8)$$
$$Key_{encrypt} = \text{SHA256}(SharedSecret \,||\, salt \,||\, \text{“Key”})$$
$$IV_{pre} = \text{SHA256}(SharedSecret \,||\, salt \,||\, \text{“IV”})$$
$$IV_{encrypt} = IV_{pre}[0:15] \oplus IV_{pre}[16:31]$$

2. Encrypts the message payload (M) with the encryption key ($Key_{encrypt}$) and IV ($IV_{encrypt}$) obtained from the previous step, using AES-256-CBC:

$$C = \text{AES} - \text{CBC}(Key_{encrypt}, IV_{encrypt}, M)$$

3. Calculates a message authentication code (MAC) of the ciphertext (C):

$$MAC_{plain} = \text{SHA256}(C)$$
$$MAC_{enc} = \text{AES} - \text{ECB}(Key_{encrypt}, MAC_{plain}[0:15] \oplus MAC_{plain}[16:31])$$

4. Sends the message to the recipient consisting of the following fields.

$version$	$content\ type$	$salt$	C	MAC	$sender\ key\ ID$	$recipient\ key\ ID$

Message fields, other than the ones generated in the process are as follows.

- **Version, Content type**: Serve to identify the Letter Sealing version used to encrypt the message
- **Sender key ID**: Used by the message recipient to retrieve the public key used in encrypting the message
- **Recipient key ID**: Helps verifying that the message can be decrypted using the current local private key

Messages targeting a previous key pair (such as one used before migrating to the current device) may not be decrypted. To facilitate device migration, LINE clients automatically request the LINE messaging server to resend recent messages targeting a previous key pair.

Message recipient

The LINE client receiving an encrypted message decrypts the message in the following steps. The receiving client:

1. Determines whether they can decrypt the message.
2. (If the client can decrypt the message) Derives the shared secret, symmetric encryption key, and IV.
3. Calculates the message authentication code (MAC) of the ciphertext received and compares the value with the MAC contained in the message. If the codes match, the client decrypts and displays the message to the LINE user. If not, the client does not display the message.

1:1 Message Encryption with Letter Sealing v2

In Letter Sealing v2, LINE encrypts each message with a unique encryption key and a random nonce.

Message sender

The sending LINE client encrypts the message in the following steps. The sending client:

1. Derives an encryption key using two values; the shared secret calculated in the client-to-client key exchange step and a 16-byte randomly generated salt.

$$salt = random_{secure}(16)$$
$$Key_{encrypt} = \text{SHA256}(SharedSecret \,||\, salt \,||\, \text{“Key”})$$

2. Calculates nonce by concatenating an 8-byte per-chat counter with a 4-byte randomly generated value.

$$nonce[12] = per\_chat_{counter}[8] \,||\, random_{secure}(4)$$

3. Encrypts the message payload (M) with AES256-GCM, using the key($Key_{encrypt}$) and $nonce$ calculated. GCM being an AEAD (Authenticated Encryption with Associated Data) scheme, ensures data confidentiality and integrity. Letter Sealing v2 adds message metadata as associated data for integrity enforcement. The metadata is as follows.

$$AAD =$$
$$recipientID \,||\, senderID \,||\, senderkeyID \,||\, recipientkeyID \,||\, version \,||\, contenttype$$

   The output, GCM tag, is 16-byte long.

$$(C, tag) = \text{AES} - \text{GCM}(Key_{encrypt}, nonce, M, AAD)$$

4. Sends the message to the recipient consisting of the following fields.

$version$	$content\ type$	$salt$	$C\|\|tag$	$nonce$	$sender\ key\ ID$	$recipient\ key\ ID$

The fields version, content type, sender key ID, and recipient key ID are similar to those included in messages with Letter Sealing v1. The difference lies in:

- Ciphertext and GCM tag sent concatenated as one chunk of data
- Nonce added as a separate chunk to the message

Message recipient

The LINE client receiving an encrypted message decrypts the message in the following steps. The receiving client:

1. Derives an encryption key using the shared secret and salt from the message.
2. Decrypts the ciphertext with AES-GCM.
3. Provides messages metadata as AAD (Additional Authenticated Data).
4. If the GCM tag received matches the tag computed during the decryption, the recipient displays the message to the LINE user. If not, the client does not display the message.

Group Message Encryption

Similar to 1:1 message encryption, the difference between the two versions of Letter Sealing for group message encryption is only in the message encryption process.

Key Generation and Registration

To implement encrypted group chats, LINE generates a shared group key and securely distributes the key to all group members. A shared group key is typically generated by the first member wishing to send a message to the group. To generate and distribute a shared group key, a LINE client:

1. Generates a new ECDH key pair. The private key of the pair serves as the group's shared key.
2. Retrieves the public keys of all group members.
3. Derives a set of symmetric encryption keys. A symmetric encryption key is calculated with the private key of the current user and the public key of a group member; this calculation is performed for all public keys retrieved from the previous step. The key derivation process is the same as 1:1 chats, as described in Key Generation and Registration and Client-to-Client Key Exchange sections.
4. Encrypts the shared group key with a symmetric encryption key derived from the previous step. This encryption is performed with all symmetric encryption keys, one by one.
5. Sends the encrypted shared group keys to LINE's messaging server.

LINE's messaging server associates the encrypted group keys with the group and returns the ID of the latest shared group key. When members join or leave the group chat, a new shared group key is generated and associated with the group.

Group Message Encryption with Letter Sealing v1

For LINE group chat members to send a message to the group using Letter Sealing v1, their LINE client first retrieves the encrypted shared group key ($Sharedkey_{group}$), decrypts the key, and caches the key locally.

To send a message, a group member derives an encryption key ($Key_{encrypt}$) and IV ($IV_{encrypt}$), using the shared group key and their own public key as input. The process is similar to the one used for 1:1 chats with Letter Sealing v1 as specified below.

$$SharedSecret_{group} = \text{ECDH}_{\text{curve25519}}(Sharedkey_{group}, key^{sender}_{public})$$
$$salt = random_{secure}(8)$$
$$Key_{encrypt} = \text{SHA256}(SharedSecret_{group} \,||\, salt \,||\, \text{“Key”})$$
$$IV_{pre} = \text{SHA256}(SharedSecret_{group} \,||\, salt \,||\, \text{“IV”})$$
$$IV_{encrypt} = IV_{pre}[0:15] \oplus IV_{pre}[16:31]$$

After obtaining required elements, LINE encrypts and formats message data for group messages as described for 1:1 messages. The only difference for group messages is that the fields recipient ID and the $recipient\ key\ ID$ are replaced with the group chat ID and the ID of the shared group key, respectively.

Group Message Encryption with Letter Sealing v2

Group message encryption with v2 computes the $Sharedkey_{group}$ as described in the Key Generation and Registration section. For LINE group chat members to send a group message with Letter Sealing v2, their LINE client first retrieves the encrypted shared group key ($Sharedkey_{group}$), decrypts the key, and caches the key locally.

To send a message, a group member derives an encryption key ($Key_{encrypt}$) using the shared group key and their own public key. The calculation is similar to that of 1:1 chats with v2, as follows.

$$SharedSecret_{group} = \text{ECDH}_{\text{curve25519}}(Sharedkey_{group}, key^{sender}_{public})$$
$$salt = random_{secure}(16)$$
$$Key_{encrypt} = \text{SHA256}(SharedSecret_{group} \,||\, salt \,||\, \text{“Key”})$$
$$nonce[12] = per\_chat_{counter}[8] \,||\, random_{secure}(4)$$

LINE encrypts and formats message data for group messages as described for 1:1 messages. The only difference for group messages is that the field $recipient\ key\ ID$ is replaced with the ID of the shared group key.

1:1 VoIP Encryption

Letter Sealing is applicable not only on messages but on VoIP calls as well. Keys for encrypting VoIP traffic are obtained using the ECDH key exchange algorithm. The curve used in LINE's VoIP encryption protocol is secp256r1[8]. LINE uses AES for symmetric encryption and HKDF[9] for deriving symmetric keys.

To start a call:

1. The caller generates a new ephemeral key pair and sends the pair to the callee as a part of the call request.
2. After the callee receives the call request, the callee generates own ephemeral key pair and sends the pair back to the caller.

User identity is guaranteed by LINE's signaling server which signs call set up messages.

$$
\begin{aligned}
&SharedSecret \\
&= \mathrm{ECDH}_{\mathrm{secp256r1}}(EphemeralKey^{caller}_{private}, EphemeralKey^{callee}_{public}) \\
&= \mathrm{ECDH}_{\mathrm{secp256r1}}(EphemeralKey^{caller}_{public}, EphemeralKey^{callee}_{private})
\end{aligned}
$$

After both parties exchange keys, each party generates a master secret and derive a VoIP session key and salt as follows:

$$
\begin{aligned}
salt_{rx} &= random_{secure}(16) \\
salt_{tx} &= random_{secure}(16) \\
MasterSecret_{rx} &= \\
\mathrm{HMAC}_{\mathrm{SHA256}}&(\mathrm{HKDF}_{\mathrm{SHA256}}(SharedSecret, EphemeralKey^{caller}_{public}, EphemeralKey^{callee}_{public}, salt_{rx})) \\
MasterSecret_{tx} &= \\
\mathrm{HMAC}_{\mathrm{SHA256}}&(\mathrm{HKDF}_{\mathrm{SHA256}}(SharedSecret, EphemeralKey^{callee}_{public}, EphemeralKey^{caller}_{public}, salt_{tx}))
\end{aligned}
$$

$MasterSecret_{rx|tx}$ and $salt_{rx|tx}$ serve as the master key and master salt used to initialize SRTP[10] and used to generate $audio|video|data\_srtp\_key_{rx|tx}$, respectively. Both audio and video media streams are encrypted using the `AES_CM_128_HMAC_SHA1_80` crypto-suite[11].

$$
\begin{aligned}
&/* \text{ Audio SRTP Key } */ \\
&audio\_srtp\_key_{rx} = \text{HMAC}_{\text{SHA256}}(\text{“AUDIO”}, MasterSecret_{rx}) \\
&audio\_srtp\_key_{tx} = \text{HMAC}_{\text{SHA256}}(\text{“AUDIO”}, MasterSecret_{tx}) \\
&/* \text{ Video SRTP Key } */ \\
&video\_srtp\_key_{rx} = \text{HMAC}_{\text{SHA256}}(\text{“VIDEO”}, MasterSecret_{rx}) \\
&video\_srtp\_key_{tx} = \text{HMAC}_{\text{SHA256}}(\text{“VIDEO”}, MasterSecret_{tx}) \\
&/* \text{ DATA SRTP Key } */ \\
&data\_srtp\_key_{rx} = \text{HMAC}_{\text{SHA256}}(\text{“DATA”}, MasterSecret_{rx}) \\
&data\_srtp\_key_{tx} = \text{HMAC}_{\text{SHA256}}(\text{“DATA”}, MasterSecret_{tx})
\end{aligned}
$$

Media Encryption

Letter Sealing is applied on media file messages (image, audio, video and file attachment) since LINE version 13.3.0 on Android and version 13.15.0 on iOS. Each media file is encrypted using a unique random symmetric key, which is end-to-end encrypted and securely shared with recipients using Letter Sealing v2 message encryption.

Key Generation and Key Exchange

The ephemeral symmetric key is generated by the sender and used to encrypt or decrypt the media files. The client follows these steps:

1. A random key material of 32 bytes is generated from a secure random source.

$$KM = random_{secure}(32)$$

2. If the media file is sent in a 1:1 chat, the key material is encrypted and shared with the recipient using the method described in 1:1 Message Encryption.
   If the media file is sent in a group chat, the key material is encrypted and shared with the recipients using the method described in Group Message Encryption.
3. The client derives the key material using HKDF-SHA256 to obtain a 32-byte encryption key, a 32-byte MAC key, and a 12-byte IV. These derived keys are used by the sender and the receivers to encrypt, decrypt and verify the integrity of the media files.

$$Key_{encrypt}[32], Key_{mac}[32], IV[12] = \text{HKDF}_{\text{SHA256}}(KM, \text{“FileEncryption”}, 76)$$

Image, Audio and File Encryption

Image, audio and file attachment media files are encrypted using AES-256-CTR. The integrity of the files follows a Encrypt-Then-Mac approach, using HMAC-SHA256.

1. The media file payload P is encrypted with the encryption key $Key_{encrypt}$ using AES-256-CTR. The IV is extended with four $0x00$ bytes, to form a 16-byte array.

$$C = \text{AES} - \text{CTR}(Key_{encrypt}, IV||\text{0x00}||\text{0x00}||\text{0x00}||\text{0x00}, P)$$

2. The MAC of the encrypted file C is calculated using HMAC-SHA256 and the MAC key Key_{mac}.

$$MAC = \text{HMAC}_{\text{SHA256}}(Key_{mac}, C)$$

3. The sender sends the ciphertext C and the MAC MAC to the server.
4. The receiver retrieves the ciphertext and the MAC from the server, then performs the inverse steps: deriving the symmetric keys from the key material, verifying the integrity of the encrypted file, and finally decrypting the file.

Video Encryption

The LINE client needs to be able to start playing videos before the entire file has been downloaded. For this reason, the encryption of video media files differs slightly from other media types, to allow verifying the integrity of the ciphertext chunks while it is being decrypted.

1. The encryption of the video is done similarly to other media types, by encrypting the file payload P with the encryption key $Key_{encrypt}$ using AES-256-CTR. The IV is extended with four $0x00$ bytes, to form a 16-byte array.

$$C = \text{AES} - \text{CTR}(Key_{encrypt}, IV||\text{0x00}||\text{0x00}||\text{0x00}||\text{0x00}, P)$$

2. The encrypted file C is split in N chunks $C_0, C_1, \ldots, C_N$ of 128 KB, which are all hashed independently using SHA256.

$$\forall i \in 0..N, H_i = \text{SHA256}(C_i)$$

3. A MAC is computed on the concatenation of the N chunk hashes H_i, using HMAC-SHA256 and the MAC key Key_{mac}.

$$MAC = \text{HMAC}_{\text{SHA256}}(Key_{mac}, H_0||H_1||\ldots||H_N)$$

4. The sender sends the ciphertext C, the MAC MAC, and the list of all chunk hashes H_i to the server.

The decryption process by the receiver occurs in a different order than the encryption process from the sender.

1. The receiver downloads the MAC MAC and the list of all chunk hashes $H_0, H_1, \ldots, H_N$ from the server.
2. A MAC MAC' is computed on the concatenation of the N chunk hashes H_i, using HMAC-SHA256 and the MAC key Key_{mac}.

   $$MAC' = \text{HMAC}_{\text{SHA256}}(Key_{mac}, H_0||H_1||\ldots||H_N)$$

   The computed MAC' is compared to the MAC received from the server to verify the integrity of the list of chunk hashes.
3. The receiver starts downloading the ciphertext C of the video media file. For each 128 KB chunk C_i, the client computes its hash H'_i, and compares it to the corresponding hash H_i received from the server.

   $$H'_i = \text{SHA256}(C_i)$$

   If the hash H'_i matches the hash H_i, the chunk can be decrypted. If the hash H'_i does not match, the client aborts the decryption process and stops playing the video.
4. The chunk C_i is decrypted with the key $Key_{encrypt}$ using AES-256-CTR. The 12-byte IV is extended with the chunk offset in the file, divided by the size of an AES block, to form a 16-byte array. Since chunks are 128 KB, this corresponds to multiplying the chunk index i by 8192.

   $$P_i = \text{AES} - \text{CTR}(Key_{encrypt}, IV||\text{to\_bytes}(i * 8192), C_i)$$

   The decrypted chunk P_i is cached for playback by the video player, and the decryption can continue with the next chunk C_{i+1}.

Conclusion

Messaging traffic between LINE clients and LINE servers is protected with forward-secure encryption, and both text messages and media streams in VoIP calls are end-to-end encrypted.

LINE's Letter Sealing protocol ensures that no third parties or LY Corporation can decrypt private calls and messages between users; encrypted communication can only be decrypted by the intended recipient.

References

[1] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2). RFC 7540, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7540.txt.

[2] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018.

[3] LY Corporation. Transparency report. https://www.lycorp.co.jp/en/privacy-security/privacy/transparency/.

[4] LY Corporation. Line encryption report. https://www.lycorp.co.jp/en/privacy-security/security/transparency/encryption-report/2024/.

[5] Takanori Isobe and Kazuhiko Minematsu. Breaking message integrity of an end-to-end encryption scheme of line. In *European Symposium on Research in Computer Security*, pages 249–268. Springer, 2018.

[6] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[7] David McGrew and John Viega. The galois/counter mode of operation (gcm). *submission to NIST Modes of Operation Process*, 20:0278–0070, 2004.

[8] Daniel RL Brown. Sec 2: Recommended elliptic curve domain parameters. *Standards for Efficient Cryptography*, 2010.

[9] H. Krawczyk and P. Eronen. Hmac-based extract-and-expand key derivation function (hkdf). RFC 5869, RFC Editor, May 2010. http://www.rfc-editor.org/rfc/rfc5869.txt.

[10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The secure real-time transport protocol (srtp). RFC 3711, RFC Editor, March 2004. http://www.rfc-editor.org/rfc/rfc3711.txt.

[11] F. Andreasen, M. Baugher, and D. Wing. Session description protocol (sdp) security descriptions for media streams. RFC 4568, RFC Editor, July 2006. http://www.rfc-editor.org/rfc/rfc4568.txt.