LINE Encryption Overview

Technical Whitepaper

September 29, 2016 Version 1.0

Copyright

This document is an intellectual property of LINE Corp.; unauthorized reproduction or distribution of this document, or any portion of it is prohibited by law.
This document is provided for informational purposes only. LINE Corp. has endeavored to verify the completeness and accuracy of information contained in this document, but it does not take the responsibility for possible errors or omissions in this document. Therefore, the responsibility for the usage of this document or the results of the usage falls entirely upon the user, and LINE Corp. does not make any explicit or implicit guarantee regarding this.
Software products or merchandises mentioned in this document, including relevant URL information, conform to the copyright laws of their respective owners. The user is solely responsible for any results occurred by not complying with applicable laws.
LINE Corp. may modify the details of this document without prior notice.

Document Information

About This Document

This document provides technical details about the encryption protocols and algorithms used in LINE's messaging and VoIP platform.

Audience

This document is intended for security engineers and developers with a strong understanding of encryption technologies.

Contact Information

If you have any questions related to this document, or find any errors, please contact us at:
LINE Security Team (dl_secwhitepaper@linecorp.com)

Revision History

Ver.	Date	Changes made
1.0	2016-09-29	Initial Publication

Table of Contents

1. Introduction

This whitepaper provides technical details about the communication protocols and encryption algorithms integrated into LINE’s messaging and VoIP platform. This document focuses on LINE’s Android and iOS mobile clients. LINE clients on other platforms might feature a slightly different implementation.

The protocols described in this document are integrated in LINE 6.7 and later.

We describe account registration, server-to-client encryption, and end-to-end encryption for messaging and VoIP.

2. Registration

2.1 Account Creation

In order to create a LINE account, users must have a valid phone number, or a Facebook account. An email address and password can optionally be added to the LINE account after registration.

Users start the account creation process by sending their phone number to LINE’s registration server. The server generates a random 4-digit PIN code and sends it to the specified number via SMS (IVR is also supported). Users verify ownership of the phone number by entering the 4-digit code into the LINE client application, which passes it on to the registration server. The server verifies the sent code and completes the registration if it matches the originally sent value. Upon successful registration, the client receives a unique user ID and an authentication key. The key is used to generate authentication tokens for all subsequent requests.

2.2 Email Address and Password Registration

Users can optionally add an email address and password to their LINE account. The email address and password are used for account migration, login from desktop LINE clients, as well as for access to LINE’s Web-based services.

When a user registers an email address and password, LINE sends a randomly generated 4-digit verification code to the specified address. Users verify ownership of the email address by entering the verification PIN code into the LINE application, or by clicking the verification link included in the verification email on their mobile device. If verification is successful, LINE authentication by email address and password is enabled for the user’s account.

3. Client-to-Server Transport Encryption

3.1 Protocol Overview

The main transport protocol used in LINE mobile clients is based on SPDY 2.0 [1]. While the SPDY protocol typically relies on TLS to establish an encrypted channel, LINE's implementation uses a lightweight handshake protocol to establish the transport keys used for application data encryption.

Our handshake protocol is loosely based on the 0-RTT handshake in TLS v1.3 [2]. LINE's transport encryption protocol uses elliptic curve cryptography (ECC) with the *secp256k1* curve [3] to implement key exchange and server identity verification. We use AES for symmetric encryption and derive symmetric keys using HKDF [4].

We describe the protocol in more detail below.

3.1.1 Static Keys

In order to guarantee that clients only connect to legitimate LINE servers, we use static ECC key pairs. LINE servers securely store the private part of each pair, while the corresponding public keys are embedded in LINE client applications.

We use two types of static keys:

- ECDH key pair for key exchange: $(\mathrm{static}_{\mathrm{private}}, \mathrm{static}_{\mathrm{public}})$
- ECDSA key pair for server identity verification: $(\mathrm{sign}_{\mathrm{private}}, \mathrm{sign}_{\mathrm{public}})$

Because clients are pre-initialized with the static ECDH key described above, clients can include encrypted application data in the first flight (0-RTT data).

3.1.2 **Handshake Protocol**

The client and server exchange the following messages in order to establish the transport key used to protect application data.

I. **Client Hello**

1. Generate an initial ephemeral ECDH key and a 16-byte client nonce.

$$(\text{c\_init}_{\text{public}}, \text{c\_init}_{\text{private}}) = \text{ECDH}_{\text{generate}}()$$
$$\text{c}_{\text{nonce}} = \text{random}_{\text{secure}}()$$

2. Derive a temporary transport key and initialization vector (IV) using the server's static key and the initial ephemeral key generated in 1. The key and IV are both 16 bytes long.

$$\text{len}_{\text{key}} = 16$$
$$\text{len}_{\text{iv}} = 16$$
$$\text{shared}_{\text{temp}} = \text{ECDH}(\text{c\_init}_{\text{private}}, \text{static}_{\text{public}})$$
$$\text{MS}_{\text{temp}} = \text{HKDF}_{\text{ex}}(\text{c}_{\text{public}}||\text{c}_{\text{nonce}}, \text{shared}_{\text{temp}})$$
$$\text{keyiv}_{\text{temp}} = \text{HKDF}_{\text{exp}}(\text{MS}_{\text{temp}}, \text{"legy temp key"}, \text{len}_{\text{key}} + \text{len}_{\text{iv}})$$
$$\text{key}_{\text{temp}} = \text{keyiv}_{\text{temp}}[0:15]$$
$$\text{iv}_{\text{temp}} = \text{keyiv}_{\text{temp}}[16:31]$$

3. Generate an ephemeral ECDH client handshake key.

$$(\text{c}_{\text{public}}, \text{c}_{\text{private}}) = \text{ECDH}_{\text{generate}}()$$

4. Encrypt c_{public} and application data with key_{temp} and iv_{temp}. (See 3.2 for details about the encryption method.).

$$\text{data}_{\text{enc}} = \text{ENC}(\text{key}_{\text{temp}}, \text{iv}_{\text{temp}}, \text{c}_{\text{public}}||\text{app data})$$

5. Send the following data to the server:

$$[\text{static}_{\text{key version}}, \text{c}_{\text{public}}, \text{c}_{\text{nonce}}, \text{data}_{\text{enc}}]$$

II. **Server Hello**

1. Calculate the temporary transport key key_{temp} and IV iv_{temp} using the server's static ECDH key and the client's initial ephemeral key.

$$\text{shared}_{\text{temp}} = \text{ECDH}(\text{static}_{\text{private}}, \text{c\_init}_{\text{public}})$$
$$\text{MS}_{\text{temp}} = \text{HKDF}_{ex}(\text{c}_{\text{public}}||\text{c}_{\text{nonce}}, \text{shared}_{\text{temp}})$$
$$\text{keyiv}_{\text{temp}} = \text{HKDF}_{\text{exp}}(\text{MS}_{\text{temp}}, \text{"legy temp key"}, \text{len}_{\text{key}} + \text{len}_{\text{iv}})$$
$$\text{key}_{\text{temp}} = \text{keyiv}_{\text{temp}}[0:15]$$
$$\text{iv}_{\text{temp}} = \text{keyiv}_{\text{temp}}[16:31]$$

2. Decrypt received application data with key_{temp} and extract c_{public}.

3. Generate an ephemeral key pair and a 16-byte server nonce.

$$(\text{s}_{\text{private}}, \text{s}_{\text{public}}) = \text{ECDH}_{\text{generate}}()$$
$$\text{s}_{\text{nonce}} = \text{random}_{\text{secure}}()$$

4. Derive the forward-secure (FS) transport key and IV.

$$\mathrm{len}_{\mathrm{key}} = 16$$

$$\mathrm{len}_{\mathrm{iv}} = 16$$

$$\mathrm{shared}_{\mathrm{FS}} = \mathrm{ECDH}\left(\mathrm{s}_{\mathrm{private}}, \mathrm{c}_{\mathrm{public}}\right)$$

$$\mathrm{MS}_{\mathrm{FS}} = \mathrm{HKDF}_{\mathrm{ex}}(\mathrm{c}_{\mathrm{nonce}}||\mathrm{s}_{\mathrm{nonce}}, \mathrm{shared}_{\mathrm{FS}})$$

$$\mathrm{keyiv}_{\mathrm{FS}} = \mathrm{HKDF}_{exp}(\mathrm{MS}_{\mathrm{FS}}, \text{"legy fs key"}, \mathrm{len}_{\mathrm{key}} + \mathrm{len}_{\mathrm{iv}})$$

$$\mathrm{key}_{\mathrm{FS}} = \mathrm{keyiv}_{\mathrm{FS}}[0\colon 15]$$

$$\mathrm{iv}_{\mathrm{FS}} = \mathrm{keyiv}_{\mathrm{FS}}[16\colon 31]$$

5. Generate and sign the handshake state using the server's static signing key.

$$\mathrm{state} = \mathrm{SHA256}(\mathrm{c}_{\mathrm{public}}||\mathrm{c}_{\mathrm{nonce}}||\mathrm{s}_{\mathrm{public}}||\mathrm{s}_{\mathrm{nonce}})$$

$$\mathrm{state}_{\mathrm{sign}} = \mathrm{ECDSA}_{\mathrm{sign}}(\mathrm{state}, \mathrm{sign}_{\mathrm{private}})$$

6. Encrypt application data with $\mathrm{key}_{\mathrm{FS}}$ and $\mathrm{iv}_{\mathrm{FS}}$.

$$\mathrm{data}_{\mathrm{enc}} = \mathrm{ENC}(\mathrm{key}_{\mathrm{FS}}, \mathrm{iv}_{\mathrm{FS}}, \text{app data})$$

7. Send the following data to client:

$$[\mathrm{s}_{\mathrm{public}}, \mathrm{s}_{\mathrm{nonce}}, \mathrm{state}_{\mathrm{sign}}, \mathrm{data}_{\mathrm{enc}}]$$

III. Client Finish

1. Verify the handshake signature. If the signature verifies, proceed to the next step. If not, abort the connection.

$$\mathrm{valid} = \mathrm{ECDSA}_{\mathrm{verify}}(\mathrm{state}_{\mathrm{sign}}, \mathrm{sign}_{\mathrm{public}})$$

2. Derive $\mathrm{key}_{\mathrm{FS}}$ and $\mathrm{iv}_{\mathrm{FS}}$.

$$\mathrm{shared}_{\mathrm{FS}} = \mathrm{ECDH}\left(\mathrm{c}_{\mathrm{private}}, \mathrm{s}_{\mathrm{public}}\right)$$

$$\mathrm{MS}_{\mathrm{FS}} = \mathrm{HKDF}_{\mathrm{ex}}(\mathrm{c}_{\mathrm{nonce}}||\mathrm{s}_{\mathrm{nonce}}, \mathrm{shared}_{\mathrm{FS}})$$

$$\mathrm{keyiv}_{\mathrm{FS}} = \mathrm{HKDF}_{\mathrm{exp}}(\mathrm{MS}_{\mathrm{FS}}, \text{"legy fs key"}, \mathrm{len}_{\mathrm{key}} + \mathrm{len}_{\mathrm{iv}})$$

$$\mathrm{key}_{\mathrm{FS}} = \mathrm{keyiv}_{\mathrm{FS}}[0\colon 15]$$

$$\mathrm{iv}_{\mathrm{FS}} = \mathrm{keyiv}_{\mathrm{FS}}[16\colon 31]$$

3. Encrypt all subsequent application data using $\mathrm{key}_{\mathrm{FS}}$ and $\mathrm{iv}_{\mathrm{FS}}$.

After the handshake is complete, both client and server share a forward-secure symmetric key $\mathrm{key}_{\mathrm{FS}}$ and can create a secure channel for application data. Application data encryption is described in the next section.

3.2 Application Data Encryption

Application data is encrypted with the 128-bit key $\mathrm{key}_{\mathrm{FS}}$ using the AES-GCM [5] AEAD cipher. Both client and server generate a unique nonce for each encryption operation. The nonce is calculated by combining a client/server marker, a 64-bit sequence number $\mathrm{num}_{\mathrm{seq}}$, and the $\mathrm{iv}_{\mathrm{FS}}$ obtained in the handshake process.

$$\mathrm{nonce} = (\mathrm{marker}\ ||\mathrm{num}_{\mathrm{pseq}}) \oplus \mathrm{iv}_{\mathrm{FS}}$$

The sequence number is reset to zero each time the encryption key changes.

Application data is encrypted using the following algorithm:

$$\mathrm{data}_{\mathrm{enc}} = \mathrm{AESGCM}(\mathrm{key}_{\mathrm{FS}}, \mathrm{nonce}, \mathrm{app\ data}\ )$$

3.3 Encryption Scope

Currently, only SPDY data frames are encrypted. Control frames in LINE's transport protocol do not carry any confidential information; they only include endpoint identifiers and message metadata.

4. Message End-to-End Encryption

4.1 Letter Sealing Overview

Letter Sealing is the common name of all end-to-end encrypted (E2EE) protocols integrated in LINE’s messaging and VoIP service. In this chapter, we focus on Letter Sealing as applied to messaging. We discuss Letter Sealing for VoIP in Chapter 5.

LINE messages are locally encrypted on each client device before being sent to LINE’s messaging server, and can only be decrypted by their intended recipient. Letter Sealing is applied only to message payloads, and message metadata (sender ID, recipient ID, and so on) is not encrypted.

The main cryptographic algorithms used in Letter Sealing for messaging are listed in the following table.

Key exchange algorithm	ECDH over Curve25519 [6]
Message encryption algorithm	AES-256 in CBC mode
Message hash function	SHA-256

4.2 1:1 Message Encryption

The following section describes Letter Sealing’s 1:1 message exchange protocol.

4.2.1 Key Generation and Registration

In order to be able to send encrypted messages, each LINE client application generates a Letter Sealing ECDH key pair, and saves it securely in the application’s private storage area. The key pair is generated when the user first launches the LINE applications or when they turn Letter Sealing back on after disabling it (Letter Sealing is enabled by default for current mobile clients).

After generating the device key pair, each LINE client registers its public key with LINE's messaging server. The server associates the key with the currently authenticated user and sends back a unique key ID to the client. Each key ID is bound to a specific user and represents the current version of that 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.

4.2.2 Client-to-Client Key Exchange

In order to be able to exchange encrypted messages, clients must share a common cryptographic secret. When a LINE client wishes to send a message, it first retrieves the current public key of the recipient. Next, the client passes its own private key and the recipient's public key to the ECDH algorithm in order to generate a shared secret. The recipient generates the same shared secret using their own private key and the sender's public key, as shown below.

$$\begin{aligned}&\text{Shared Secret}\\&= \mathrm{ECDH}_{\mathrm{curve25519}}\left(\mathrm{key}_{\mathrm{private}}^{\mathrm{user1}}, \mathrm{key}_{\mathrm{public}}^{\mathrm{user2}}\right)\\&= \mathrm{ECDH}_{\mathrm{curve25519}}(\mathrm{key}_{\mathrm{private}}^{\mathrm{user2}}, \mathrm{key}_{\mathrm{public}}^{\mathrm{user1}})\end{aligned}$$

The above process is transparent to users. Users who want to make sure they are communicating with the expected recipient can display the recipient's public key fingerprint and verify it out-of-band.

4.2.3 Message Encryption

LINE encrypts each message with a unique encryption key and IV. The encryption key and IV are derived from the shared secret calculated in 4.2.2, and a randomly generated 8-byte salt as follows:

$$\mathrm{Key}_{\mathrm{encrypt}} = \mathrm{SHA256}(\text{Shared Secret} || \text{ salt } || \text{ "Key" })$$

$$\mathrm{IV}_{\mathrm{pre}} = \mathrm{SHA256}(\text{Shared Secret} || \text{ salt } || \text{ "IV" })$$

$$\mathrm{IV}_{\mathrm{encrypt}} = \mathrm{IV}_{\mathrm{pre}}[0:15] \oplus \mathrm{IV}_{\mathrm{pre}}[16:31]$$

The generated key and IV are used to encrypt the message payload M using 256-bit AES in CBC block mode.

$$\mathrm{C} = \mathrm{AESCBC}(\mathrm{Key}_{\mathrm{encrypt}}, \mathrm{IV}_{\mathrm{encrypt}}, \mathrm{M})$$

Next, LINE calculates a message authentication code (MAC) of the ciphertext C, as follows:

$$\mathrm{MAC}_{\mathrm{plain}} = \mathrm{SHA256}(\mathrm{C})$$

$$\mathrm{MAC}_{\mathrm{enc}} = \mathrm{AESECB}(\mathrm{Key}_{\mathrm{encrypt}}, \mathrm{MAC}_{\mathrm{plain}}[0:15] \oplus \mathrm{MAC}_{\mathrm{plain}}[16:31])$$

Finally, the following data is included in the message sent to the recipient:

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

The version and content type fields serve to identity the Letter Sealing version used to create the message. Recipients use the sender key ID to retrieve the public key used to encrypt the message. The recipient key ID value helps verify that the message can be decrypted using the current local private key. Messages that target a previous key pair (such as one used before migrating to the current device) cannot be decrypted. To facilitate device migration, LINE clients automatically request recent messages targeting a previous key pair to be resent.

Once the recipient determines that they can decrypt a message, they derive the shared secret, symmetric encryption key, and IV as described above. Next, LINE calculates the MAC of the received ciphertext, and compares it with the MAC value included in the message. If they match, the contents of the message is decrypted and displayed. Otherwise, the message is discarded.

4.3 1:N (group) Message Encryption

In order to implement 1:N encrypted chats, LINE generates a shared group key $\mathrm{Sharedkey}_{\mathrm{group}}$, which is then securely distributed to all group members. The group key is typically generated by the first member that wants to send a message to the group.

To associate a group key with a group, LINE first generates a new ECDH key pair. The private part serves as the group's shared key. LINE then retrieves the public keys of all current group members, and calculates a set of symmetric encryption keys using the current user's private key and each group member's public key. The key derivation process is the same for 1:1 chats, as described in 4.2.1 and 4.2.2. Next, the group shared key is encrypted individually with each of the generated symmetric keys, and the encrypted data is sent to the messaging server. The server associates the encrypted group keys with the group and returns the current shared key ID.

When members join or leave the group, a new group shared key is generated and associated with the group.

When group members want to send a message to the group, they first retrieve the encrypted $\mathrm{Sharedkey}_{\mathrm{group}}$, decrypt it, and cache it locally. To send a message, each member derives an encryption key and IV, using the group's shared key and their own public key as input. The process is similar to the one used for 1:1 chats, and is presented below.

$$\mathrm{Shared\ Secret}_{\mathrm{group}} = \mathrm{ECDH}_{\mathrm{curve25519}}\left(\mathrm{Sharedkey}_{\mathrm{group}}, \mathrm{key}_{\mathrm{public}}^{\mathrm{sender}}\right)$$
$$\mathrm{Key}_{\mathrm{encrypt}} = \mathrm{SHA256}(\mathrm{Shared\ Secret}_{\mathrm{group}} \,||\, \mathrm{salt} \,||\, \text{"Key"})$$
$$\mathrm{IV}_{\mathrm{pre}} = \mathrm{SHA256}(\mathrm{Shared\ Secret}_{\mathrm{group}} \,||\, \mathrm{salt} \,||\, \text{"IV"})$$
$$\mathrm{IV}_{\mathrm{encrypt}} = \mathrm{IV}_{\mathrm{pre}}[0:15] \oplus \mathrm{IV}_{\mathrm{pre}}[16:31]$$

Message data is encrypted and formatted as described in 4.2.3, with the only difference that the recipient key ID field is replaced with the key ID of the group’s shared key.

4.4 Encryption Scope

Letter Sealing is currently applied to text messages and location messages.

5. VoIP End-to-End Encryption

In addition to message encryption, LINE also supports end-to-end encryption for free VoIP calls. Keys for VoIP traffic encryption are established using the ECDH key exchange algorithm. The curve used in LINE's VoIP encryption protocol is *secp256r1* [3].

To start a call, the caller generates a new ephemeral key pair and sends it to the callee as part of the call request. After the callee receives the call request, they generate their own ephemeral key pair and send it back to the caller. User identity is guaranteed by the signaling server which signs call setup messages with a static key whose public part is embedded in LINE clients.

After both parties exchange keys, they generate a master secret, and derive a VoIP session key and salt as follows:

$$\begin{aligned}&\mathrm{Secret_{Master}}\\ &= \mathrm{ECDH_{secp256r1}}\left(\mathrm{Ephemeral\ key}_{\mathrm{private}}^{\mathrm{caller}}, \mathrm{Ephemeral\ key}_{\mathrm{public}}^{\mathrm{callee}}\right)\\ &= \mathrm{ECDH_{secp256r1}}\left(\mathrm{Ephemeral\ key}_{\mathrm{public}}^{\mathrm{caller}}, \mathrm{Ephemeral\ key}_{\mathrm{private}}^{\mathrm{callee}}\right)\end{aligned}$$

$$\mathrm{Key_{VoIP}} = \mathrm{HMAC_{SHA512}}(\mathrm{Secret_{Master}}, \mathrm{Key_{Call}})[0{:}\,15]$$
$$\mathrm{Salt_{VoIP}} = \mathrm{HMAC_{SHA512}}(\mathrm{Secret_{Master}}, \mathrm{Key_{Call}})[16{:}\,29]$$

Here $\mathrm{Key_{Call}}$ is a unique call ID, randomly generated at call initiation. $\mathrm{Key_{VoIP}}$ and $\mathrm{Salt_{VoIP}}$ serve as the master key and master salt used to initialize SRTP [7], respectively. Both audio and video media streams are encrypted using the `AES_CM_128_HMAC_SHA1_80` crypto-suite [8].

6. Conclusion

Messaging traffic between LINE clients and our servers is protected with forward-secure encryption, and both text messages and media streams in VoIP calls are end-to-end encrypted. Our end-to-end encryption protocols ensure that neither third parties, nor LINE Corporation can decrypt private calls and messages between users; encrypted communication can only be decrypted by the intended recipient.

7. References

[1] M. Belshe, R. Peon, et al., "SPDY Protocol - Draft 2", https://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2

[2] E. Rescorla, "Transport Layer Security (TLS) Protocol Version 1.3", https://tools.ietf.org/html/draft-ietf-tls-tls13-16

[3] Standards for Efficient Cryptography Group, "SEC 2: Recommended Elliptic Curve Domain Parameters", January 2010. Version 2.0, http://www.secg.org/sec2-v2.pdf

[4] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010, http://www.rfc-editor.org/info/rfc5869

[5] D. McGrew and J. Viega., "The Galois/Counter Mode of Operation (GCM)". Manuscript, May 2005, Available from the NIST website.

[6] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records", Proceedings of PKC 2006, February 2006

[7] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004, http://www.rfc-editor.org/info/rfc3711

[8] F. Andreasen, M. Baugher, D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams ", RFC4568, July 2006, https://www.rfc-editor.org/info/rfc4568