LINE Encryption Overview

Technical Whitepaper

October 28, 2019

Version 2.0

LINE

Copyright

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
2.0	2019-10-28	Update for Letter Sealing version 2

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. Version 2 of Letter Sealing is integrated since LINE 8.15 for iOS mobile clients, since LINE 8.17 for Android mobile clients, and since version 2.6 for LINE Lite clients.

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 and register an account password. An email address 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 6-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 6-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

A password is required when creating a new LINE account. Users can optionally add an email address 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: $(\text{static}_{\text{private}}, \text{static}_{\text{public}})$
- ECDSA key pair for server identity verification: $(\text{sign}_{\text{private}}, \text{sign}_{\text{public}})$

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.

$$(\text{c\_init}_{\text{public}}, \text{c\_init}_{\text{private}}) = \text{ECDH}_{\text{generate}}()$$
$$\text{c}_{\text{nonce}} = \text{random}_{\text{secure}}(16)$$

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.

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

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:

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

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.

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

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

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

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

$$s_{nonce} = random_{secure}(16)$$

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

$$len_{key} = 16$$
$$len_{iv} = 16$$
$$shared_{FS} = ECDH(s_{private}, c_{public})$$
$$MS_{FS} = HKDF_{extract}(c_{nonce}||s_{nonce}, shared_{FS})$$
$$keyiv_{FS} = HKDF_{expand}(MS_{FS}, \text{"legy fs key"}, len_{key} + len_{iv})$$
$$key_{FS} = keyiv_{FS}[0:15]$$
$$iv_{FS} = keyiv_{FS}[16:31]$$

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

$$state = SHA256(c_{public}||c_{nonce}||s_{public}||s_{nonce})$$
$$state_{sign} = ECDSA_{sign}(state, sign_{private})$$

6. Encrypt application data with key_{FS} and iv_{FS}.

$$data_{enc} = ENC(key_{FS}, iv_{FS}, \text{app data})$$

7. Send the following data to client:

$$[s_{public}, s_{nonce}, state_{sign}, data_{enc}]$$

III. Client Finish

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

$$valid = ECDSA_{verify}(state_{sign}, sign_{public})$$

2. Derive key_{FS} and iv_{FS}.

$$shared_{FS} = ECDH(c_{private}, s_{public})$$
$$MS_{FS} = HKDF_{extract}(c_{nonce}||s_{nonce}, shared_{FS})$$
$$keyiv_{FS} = HKDF_{expand}(MS_{FS}, \text{"legy fs key"}, len_{key} + len_{iv})$$
$$key_{FS} = keyiv_{FS}[0:15]$$
$$iv_{FS} = keyiv_{FS}[16:31]$$

3. Encrypt all subsequent application data using key_{FS} and iv_{FS}.

After the handshake is complete, both client and server share a forward-secure symmetric key key_{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 key_{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 num_{seq}, and the iv_{FS} obtained in the handshake process.

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

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

Application data is encrypted using the following algorithm:

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

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.

Several message data integrity issues were discovered in the first version of Letter Sealing [6]. In order to solve these issues, Letter Sealing has been updated to version 2, which guarantees stronger protection of messages. The new version of Letter Sealing also adds integrity of message metadata.

Letter Sealing version 2 is used by default for 1:1 and 1:N messages when the sender's client supports it. If the recipient of a message does not support version 2, the protocol is downgraded to Letter Sealing version 1, and the message is sent again.

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

	Version 1	Version 2
Key exchange algorithm	ECDH over Curve25519 [7]	
Message encryption algorithm	AES-256 in CBC mode	AES-256 in GCM mode
Message hash function	SHA-256	N/A

Data authentication	AES-ECB with SHA256 MAC	AES-256 in GCM mode
Message data	Encryption and integrity	
Message metadata	Not protected	Integrity

4.2 1:1 Message Encryption

The following section describes Letter Sealing's 1:1 message exchange protocol. Key generation, registration, and client-to-client key exchange are similar between the two versions of Letter Sealing. The message encryption protocols are different and are detailed separately.

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.

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

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 (version 1)

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:

$$\text{salt} = \text{random}_{\text{secure}}(8)$$
$$\text{Key}_{\text{encrypt}} = \text{SHA256}(\text{Shared Secret}|| \text{ salt } || \text{ "Key" })$$
$$\text{IV}_{\text{pre}} = \text{SHA256}(\text{Shared Secret}|| \text{ salt } || \text{ "IV" })$$
$$\text{IV}_{\text{encrypt}} = \text{IV}_{\text{pre}}[0:15] \oplus \text{IV}_{\text{pre}}[16:31]$$

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

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

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

$$\text{MAC}_{\text{plain}} = \text{SHA256}(\text{C})$$
$$\text{MAC}_{\text{enc}} = \text{AESECB}(\text{Key}_{\text{encrypt}}, \text{MAC}_{\text{plain}}[0:15] \oplus \text{MAC}_{\text{plain}}[16:31])$$

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.

4.2.4 Message Encryption (version 2)

When using Letter Sealing v2, LINE also encrypts each message with a unique encryption key and a random nonce. The encryption key is derived from the shared secret calculated from 4.2.2, and a 16-byte randomly generated salt. The nonce consists of an 8-byte per-chat counter concatenated with a 4-byte randomly generated value. The process is as follows:

$$\text{salt} = \text{random}_{\text{secure}}(16)$$

$$\mathrm{Key}_{\mathrm{encrypt}} = \mathrm{SHA256}(\text{Shared Secret} \,||\, \text{salt} \,||\, \text{"Key"})$$
$$\mathrm{nonce}[12] = \mathrm{per\_chat}_{\mathrm{counter}}[8] \,||\, \mathrm{random}_{\mathrm{secure}}(4)$$

The generated key and the nonce are used to encrypt the message payload M using 256-bit AES in GCM mode. As GCM is an authenticated encryption mode with associated data, it can ensure data confidentiality and integrity. The message data is encrypted and authenticated, and metadata are added as associated data for integrity enforcement. Metadata are the recipient ID, the sender ID, the sender key ID, the recipient key ID, the Letter Sealing version, and the content type. The output authentication tag is 16-byte long. The encryption process is as follows:

$$\mathrm{AAD} = \text{receipient ID} \,||\, \text{sender ID} \,||\, \text{sender key ID} \,||\, \text{recipient key ID} \,||\, \text{version} \,||\, \text{content type}$$
$$(\mathrm{C}, \mathrm{tag}) = \mathrm{AESGCM}(\mathrm{Key}_{\mathrm{encrypt}}, \mathrm{nonce}, \mathrm{M}, \mathrm{AAD})$$

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

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

The version, content type, the sender key ID and the recipient key ID are similar to those included in version 1 messages. The difference comes from the ciphertext and the GCM tag which are sent concatenated as one chunk of data, and the nonce which is added as a separate chunk to the message.

Once recipients proceed to the decryption of the whole message, they first derive the encryption key using their shared secret and the salt from the message. Then, they proceed to decrypt the ciphertext with AES GCM mode, and provide the metadata of the message as additional authenticated data. If the tag received from the sender matches the tag computed during the decryption, the message can be displayed. Otherwise, the message is discarded.

4.3 1:N (group) Message Encryption

Similarly to 1:1 messages, the difference between the two versions of Letter Sealing is only in the message encryption process.

4.3.1 Key Generation and Registration

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.

4.3.2 Message Encryption (version 1)

When group members want to send a message to the group, they first retrieve the encrypted $\text{Sharedkey}_{\text{group}}$, 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 with version 1 and is presented below.

$$\text{Shared Secret}_{\text{group}} = \text{ECDH}_{\text{curve25519}}\left(\text{Sharedkey}_{\text{group}}, \text{key}_{\text{public}}^{\text{sender}}\right)$$
$$\text{salt} = \text{random}_{\text{secure}}(8)$$
$$\text{Key}_{\text{encrypt}} = \text{SHA256}(\text{Shared Secret}_{\text{group}} || \text{ salt } || \text{ "Key" })$$
$$\text{IV}_{\text{pre}} = \text{SHA256}(\text{Shared Secret}_{\text{group}} || \text{ salt } || \text{ "IV" })$$
$$\text{IV}_{\text{encrypt}} = \text{IV}_{\text{pre}}[0{:}15] \oplus \text{IV}_{\text{pre}}[16{:}31]$$

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.

4.3.3 Message Encryption (version 2)

For 1:N encryption with version 2, the $\text{Sharedkey}_{\text{group}}$ is computed as described in 4.3.1. When group members want to send a message to the group, they retrieve the $\text{Sharedkey}_{\text{group}}$ from the server, decrypt it, and cache it locally. To send a message, a group member derives an encryption key by using the group shared key, and their own public key. The process is similar to 1:1 chats with the version 2 and is presented below.

$$\text{Shared Secret}_{\text{group}} = \text{ECDH}_{\text{curve25519}}\left(\text{Sharedkey}_{\text{group}}, \text{key}_{\text{public}}^{\text{sender}}\right)$$
$$\text{salt} = \text{random}_{\text{secure}}(16)$$
$$\text{Key}_{\text{encrypt}} = \text{SHA256}(\text{Shared Secret}_{\text{group}} || \text{ salt } || \text{ "Key" })$$
$$\text{nonce}[12] = \text{per\_chat}_{\text{counter}}[8] || \text{random}_{\text{secure}}(4)$$

Message data is encrypted and formatted as described in 4.2.4, with the 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:

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

$$\begin{aligned}\mathrm{Key}_{\mathrm{VoIP}}&=\mathrm{HMAC}_{\mathrm{SHA512}}(\mathrm{Secret}_{\mathrm{Master}},\mathrm{Key}_{\mathrm{Call}})[0:15]\\\mathrm{Salt}_{\mathrm{VoIP}}&=\mathrm{HMAC}_{\mathrm{SHA512}}(\mathrm{Secret}_{\mathrm{Master}},\mathrm{Key}_{\mathrm{Call}})[16:29]\end{aligned}$$

Here $\mathrm{Key}_{\mathrm{Call}}$ is a unique call ID, randomly generated at call initiation. $\mathrm{Key}_{\mathrm{VoIP}}$ and $\mathrm{Salt}_{\mathrm{VoIP}}$ serve as the master key and master salt used to initialize SRTP[8], respectively. Both audio and video media streams are encrypted using the `AES_CM_128_HMAC_SHA1_80` crypto-suite[9].

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] Isobe T., Minematsu K., "Breaking Message Integrity of an End-to-End Encryption Scheme of LINE.", Cryptology ePrint Archive: Report 2018/668

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

[8] 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

[9] 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