

Threema Cryptography Whitepaper

Threema uses modern cryptography based on open source components that strike an optimal balance between security, performance and message size. In this whitepaper, the algorithms and design decisions behind the cryptography in Threema are explained.

Contents

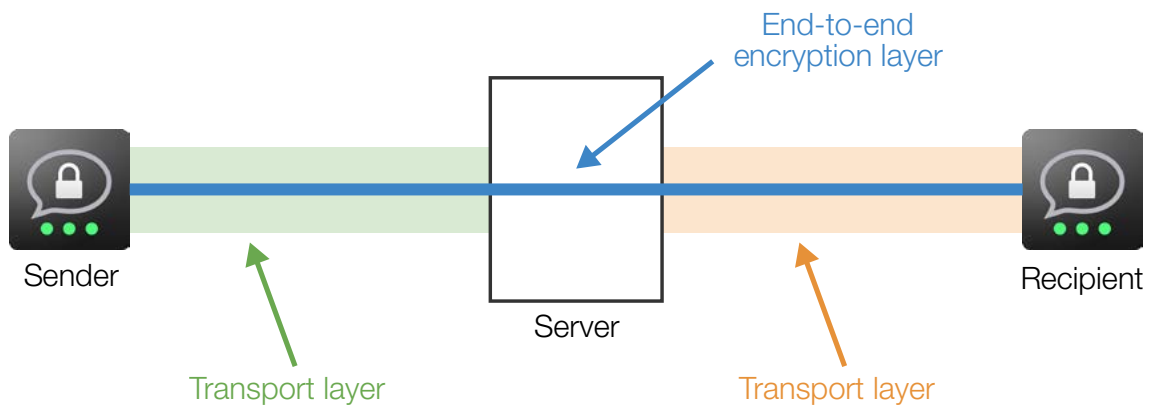
Overview	2
End-to-End Encryption	3
Key Generation and Registration	3
Key Distribution and Trust	3
Message Encryption	4
Group Messaging	4
Key Backup	5
Client-Server Protocol Description	5
Chat Protocol (Message Transport Layer)	6
Directory Access Protocol	6
Media Access Protocol	6
Cryptography Details	7
Key Lengths	7
Random Number Generation	8
Forward Secrecy	8
Non-Repudiation	9
Padding	9
Local Data Encryption	9
iOS	9
Android	9
Windows Phone	10
Key Storage	10
iOS	10
Android	10
Windows Phone	10
Address Book Synchronization	11
Linking	11
ID Revocation	12
An Example	12

Overview

Threema uses two different encryption layers to protect messages between the sender and the recipient.

- **End-to-end encryption layer:** this layer is between the sender and the recipient.
- **Transport layer:** each end-to-end encrypted message is encrypted again for transport between the client and the server, in order to protect the header information.

These layers and other important aspects of the cryptography in Threema are described in detail in the following chapters. The crucial part is that the end-to-end encryption layer passes through the server uninterrupted; the server cannot remove the inner encryption layer.



End-to-End Encryption

In Threema, all messages (whether they are simple text messages, or contain media like images, videos or audio recordings) are end-to-end encrypted. For this purpose, each Threema user has a unique asymmetric key pair consisting of a public and a private key based on Elliptic Curve Cryptography. These two keys are mathematically related, but it is not technically feasible to calculate the private key given only the public key.

Key Generation and Registration

When a Threema user sets up the app for the first time, the following process is performed:




1. The app generates a new key pair by choosing a private key at random¹, storing it securely on the device, and calculating the corresponding public key over the Elliptic Curve (Curve25519).
2. The app sends the public key to the server.
3. The server stores the public key and assigns a new random Threema ID, consisting of 8 characters out of A-Z/0-9.
4. The app stores the received Threema ID along with the public and private key in secure storage on the device.

Key Distribution and Trust

The public key of each user is stored on the directory server, along with its permanently assigned Threema ID. Any user may obtain the public key for a given Threema ID by querying the directory server. The following input values can be used for this query:

- a full 8-character Threema ID
- a hash of a E.164 mobile phone number that is linked with a Threema ID
 - see section “Address Book Synchronization” for details on the hashing
- a hash of an email address that is linked with a Threema ID

The Threema app maintains a “verification level” indicator for each contact that it has stored. The following three levels are possible:

-  Red (level 1): The public key has been obtained from the server because a message has been received from this contact for the first time, or the ID has been entered manually. No matching contact was found in the user’s address book (by phone number or email), and therefore the user cannot be sure that the person is who they claim to be in their messages.
-  Orange (level 2): The ID has been matched with a contact in the user’s address book (by phone number or email). Since the server verifies phone numbers and email addresses, the user can be reasonably sure that the person is who they claim to be.
-  Green (level 3): The user has personally verified the ID and public key of the contact by scanning their 2D code. Assuming the contact’s device has not been hijacked, the user can be very sure that messages from this contact were really written by the person that they indicate.

To upgrade a contact from the red or orange to the green level, the user must scan that contact’s public QR code. This QR code is displayed in the “My ID” section of the app, and uses the following format:

```
3mid:<identity>,<publicKeyHex>
```

where <identity> is the 8-character Threema ID, and <publicKeyHex> is the hexadecimal (lowercase) representation of the user’s 32 byte public key. The app verifies that the scanned public key matches the one that was returned by the directory server.

1 including entropy generated by user interaction; see section “Random Number Generation” for details

Message Encryption

Threema uses the “Box” model of the [NaCl Networking and Cryptography Library](#) to encrypt and authenticate messages. For the purpose of this description, assume that Alice wants to send a message to Bob. Encryption of the message using NaCl works as follows:

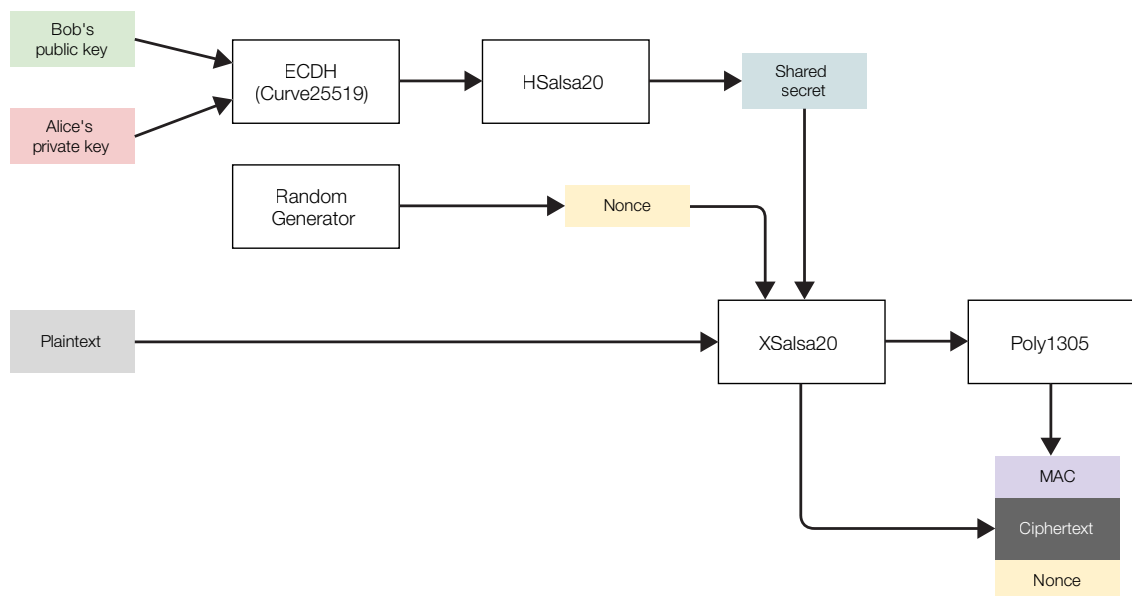
Preconditions

1. Alice and Bob have each generated a key pair consisting of a public and a private key.
2. Alice has obtained the public key from Bob over an authenticated channel.

Procedure to encrypt a message

1. Alice uses Elliptic Curve Diffie-Hellman (ECDH) over the curve Curve25519 and hashes the result with HSalsa20 to derive a shared secret from her own private key and Bob’s public key.
Note that due to the properties of the elliptic curve, this shared secret is the same if the keys are swapped (i.e. if Bob’s private key and Alice’s public key are used instead).
2. Alice generates a random nonce.
3. Alice uses the XSalsa20 stream cipher with the shared secret as the key and the random nonce to encrypt the plaintext.
4. Alice uses Poly1305 to compute a Message Authentication Code (MAC), and prepends it to the ciphertext. A portion of the key stream from XSalsa20 is used to form the MAC key.
5. Alice sends the MAC, ciphertext and nonce to Bob.

By reversing the above steps and using his own private key and the Alice’s public key, Bob can decrypt the message and verify its authenticity.



For further details, see [Cryptography in NaCl](#).

Group Messaging

In Threema, groups are managed without any involvement of the servers. That is, the servers do not know which groups exist and which users are members of which groups. When a user sends a message to a group, it is individually encrypted and sent to each other group member. This may appear wasteful, but given typical message sizes of 100-300 bytes, the extra traffic is insignificant. Media files (images, video, audio) are encrypted with a random symmetric key and uploaded only once. The same key, along with a reference to the uploaded file, is then distributed to all members of the group.

Chat Protocol (Message Transport Layer)

The chat protocol is used to transport incoming and outgoing messages between the client (app) and the Threema servers. It is a custom binary protocol that uses the NaCl library to secure the connection on the application layer. Its properties are:

- Provides Forward Secrecy
 - New temporary keys are generated whenever the app restarts
 - Temporary private keys are never stored in permanent storage, but are only kept in volatile memory
- Optimized for minimal overhead
 - Small message headers
 - Minimum number of round-trips required for connection setup
- User is authenticated using his public key during connection setup
 - Ensures that a user can only log in if he is in possession of the private key for the Threema ID

Directory Access Protocol

The directory access protocol is used for the following purposes:

- Creating new Threema IDs
- Fetching the public key of another user
- Linking email addresses and mobile phone numbers
- Matching address book contacts (hashes of phone numbers and email addresses) to Threema IDs

This protocol is based on HTTPS (HTTP with TLS). Strong TLS cipher suites with forward secrecy (ECDHE/DHE) and TLS v1.2 are supported. The servers use certificates issued by a Threema internal Certificate Authority (CA), whose CA certificate is hardcoded into the app (certificate pinning)². This precludes man-in-the-middle (MITM) attacks even if the system's trusted CA store has been tampered with, or a trusted CA has illegally issued a certificate for a Threema domain name.

Requests that access public information (i.e. fetching the public key of another user or matching address book contacts) are anonymous and do not use authentication. For requests that need to be authenticated (i.e. linking email addresses and phone numbers), a challenge/response protocol based on the user's key pair is used.

Media Access Protocol

The media servers are used for temporary storage of large media data (e.g. images, videos, audio recordings). Such media is not sent directly via the chat protocol. Instead, the following procedure is used:

1. The sender encrypts the media file with a random 256-bit symmetric key using XSalsa20 and adds a Poly1305 authenticator.
2. The sender uploads the encrypted media file to a media server via HTTPS.
3. The media server assigns a unique ID for this upload and returns it to the sender.
4. The sender sends an end-to-end encrypted message to the recipient, which contains the media ID and the symmetric key.
5. The recipient receives and decrypts the end-to-end encrypted message, obtaining the media ID and the symmetric key.
6. The recipient uses the media ID to download the encrypted media file from the media server.
7. The recipient decrypts the media file using the symmetric key.
8. The media server deletes the file upon successful download.

² Windows Phone 8.0 does not support certificate pinning. Therefore, regular commercial certificates are used to access the directory servers on Windows Phone.

The media servers use the same TLS configuration (cipher suites with forward secrecy, TLS v1.2) as the directory servers, and the app uses certificate pinning when accessing them.

Cryptography Details

As mentioned earlier, Threema uses the [NaCl Networking and Cryptography Library](#) for both the end-to-end encryption, and to secure the chat protocol at the transport level. This library uses a collection of algorithms to provide a simplified interface for protecting a message using what the authors call “public-key authenticated encryption” against eavesdropping, spoofing and tampering. By default, and as implemented in Threema, it uses the following algorithms:

- Key derivation: Elliptic Curve Diffie-Hellman (ECDH) over the curve [Curve25519](#)
- Symmetric encryption: XSalsa20
- Authentication and integrity protection: Poly1305-AES

It is worth mentioning that the ECC curve used by NaCl (and thus by Threema) is **not one of the NIST-recommended curves** that have been suspected of containing deliberately selected weakening constants in their specification.

For more details, see [Cryptography in NaCl](#).

The following implementations of the cryptographic algorithms are used:

	Curve25519	XSalsa20	Poly1305
iOS	<i>Donna</i> C implementation	reference C implementation	“53” C implementation
Android	reference C implementation (native code, fallback to jnacl)		
Windows Phone	Chaos.NaCl C# implementation		

Key Lengths

The asymmetric keys used in Threema have a length of 256 bits, and their effective ECC strength is 255 bits.

The shared secrets, which are used as symmetric keys for end-to-end message encryption (derived from the sender’s private key and the recipient’s public key using ECDH, and combined with a 192 bit nonce), have a length of 256 bits.

The random symmetric keys used for media encryption are also 256 bits long.

The message authentication code (MAC) that is added to each message to detect tampering and forgery has a length of 128 bits.

Discussion of reasonable key lengths

According to [NIST Special Publication 800-57](#) (page 64), the security level of ECC based encryption at 255 bits can be compared to RSA at roughly 2048 to 3072 bits, or a symmetric security level of ~128 bits. The possibility of a successful brute force attack on a key with a 128 bit security level is considered extremely unlikely using current technology and knowledge, according to the judgment of reputable security researchers. A revolutionary breakthrough in mathematics or quantum computing would most possibly render keys breakable anyway, whether they are 128 or 256 bits in length.

- <http://cr.yp.to/talks/2005.09.19/slides.pdf> (Daniel J. Bernstein, author of Curve25519)
- <https://www.imperialviolet.org/2014/05/25/strengthmatching.html> (Adam Langley, Google security researcher)

Random Number Generation

Threema uses random numbers for the following purposes, listed by descending order of randomness quality required:

- Private key generation
- Symmetric encryption of media files
- Nonces
- Backup encryption salt
- Padding amount determination

Nonces and salts must never repeat, but they are not required to be hard to guess.

To obtain random numbers, Threema uses the system-provided random number generator (RNG) intended for cryptographically-secure purposes provided by the device's operating system. The exact implementation varies among operating systems:

Platform	Facility	Implementation
iOS	/dev/random	Yarrow (SHA1)
Android	/dev/random ³	Linux PRNG
Windows Phone	RNGCryptoServiceProvider	unknown

Note that Threema does not use the flawed [Dual_EC_DRBG](#) random number generator.

User generated entropy for private key generation

Due to the requirement for very high quality randomness when generating the long-term private key, the user is prompted to generate additional entropy by moving a finger on the screen. The movements (consisting of coordinate values and high-resolution timestamps) are continuously collected and hashed for several seconds. The resulting entropy is then mixed (XOR) with entropy obtained from the system's RNG.

Forward Secrecy

Due to the inherently asynchronous nature of mobile messengers, providing reliable Forward Secrecy on the end-to-end layer is difficult. Key negotiation for a new chat session would require the other party to be online before the first message can be sent. Experimental schemes like caching pre-generated temporary keys from the clients on the servers increase the server and protocol complexity, leading to lower reliability and more potential for mistakes that impact security. The user experience can also be diminished by events that are not under the control of the sender, for example when the recipient loses their phone's data, and along with it the ephemeral keys. Due to these and the following considerations, Threema has implemented Forward Secrecy on the transport layer only:

- Reliability is very important to ensure that users do not feel negatively impacted by the cryptography.
- The risk of eavesdropping on any path through the Internet between the sender and the server, or between the server and the recipient, is orders of magnitude greater than the risk of eavesdropping on the server itself.

With Threema's implementation, an attacker who has captured and stored the network traffic between a client and a Threema server will not be able to decrypt it even if he learns the private key of the server or the client afterwards.

³ To avoid the flawed SecureRandom implementation in some Android versions, Threema uses its own implementation that directly accesses /dev/random (which is not affected by the SecureRandom implementation bug).

Non-Repudiation

In general, cryptographically signed messages also provide non-repudiation; i.e. the sender cannot deny having sent the message after it has been received. The NaCl library's box model uses so-called *public-key authenticators* instead, which guarantee repudiability (see <http://nacl.cr.yp.to/box.html>, "Security model"). Any recipient can forge a message that will look just like it was actually generated by the purported sender, so the recipient cannot convince a third party that the message was really generated by the sender and not forged by the recipient. However, the recipient is still protected against forgeries by third parties. The reason is that in order to perform such a forgery, the private key of the recipient is needed. Since the recipient himself will know whether or not he has used his own private key for such a forgery, he can be sure that no third party could have forged the message.

Padding

In order to thwart attempts to guess the content of short messages by looking at the amount of data, Threema adds a random amount of PKCS#7 padding to each message before end-to-end encryption.

Local Data Encryption

The Threema app stores local data (such as the history of incoming and outgoing messages, and the contact list) in encrypted form on the device. The way in which this data is encrypted varies among platforms.

iOS

On iOS, Threema stores local data in a Core Data database, which is backed by files in the app's private data directory. The iOS sandbox model ensures that no other apps can access this data directory. All files stored by Threema are protected using the iOS Data Protection feature with the `NSFileProtectionComplete` class (see [iOS Security Whitepaper](#), page 10). This means that they are only accessible while the device is unlocked, i.e. the passcode has been entered by the user. The encryption key used to protect the files is derived from the device's UID key and the user's passcode. A longer passcode will provide better security (on devices equipped with a Touch ID sensor, longer passcodes can be used without a loss of comfort as the passcode only needs to be entered after a reboot). As soon as the device is locked, the key used to protect the files is discarded and the files are inaccessible until the user unlocks the device again. This is also the reason why Threema does not perform background processing of push messages.

Note that the private key for the Threema ID is stored separately in the iOS Keychain.

Android

On Android, Threema stores local data in an SQLite database within the app's private data directory. Android ensures that no other apps can access this data directory (as long as the device is not rooted). The database itself is protected by SQLCipher with AES-256, and any media files (which are stored separately in the app's private directory within the file system) are also encrypted using AES. The key is randomly generated when the database is first created, and can be protected with a user-supplied passphrase (which is of course necessary if the user wishes to take advantage of this encryption). The passphrase is never written to permanent storage and therefore needs to be entered whenever the app process restarts (e.g. after a low-memory situation, or after the device has rebooted). Alternatively, the user may enable full-device encryption if supported by the device and Android version.

Windows Phone

On Windows Phone, Threema stores local data using SQL Server Compact Edition. The database is protected with a password, which is derived from the private key associated with the Threema ID. The private key is in turn protected using DPAPI and an optional user passphrase (see “Key Storage” for details).

Key Storage

On the user’s mobile device, the private key is stored in such a way as to prevent access by other apps on the same device, or by unauthorized users. The procedure differs between platforms.

iOS

- The private key is stored in the iOS Keychain with the attribute **kSecAttrAccessibleWhenUnlockedThisDeviceOnly**.
- Only the Threema app can access this keychain entry.
- While the entire iOS keychain is included in backups via iTunes or iCloud, the kSecAttrAccessibleWhenUnlockedThisDeviceOnly attribute causes the keychain entry of Threema to be encrypted with a device-specific key (“UID key”) that cannot be read directly and is not known to Apple. Therefore, the keychain entry is only usable if the iTunes/iCloud backup is restored to the same device.
 - See [iOS Security Whitepaper](#), page 8
- When the Threema app is deleted and reinstalled, the keychain entry persists and the Threema ID is not lost.

Android

- The private key is stored in a file in the app’s private data directory.
- Other apps cannot access this file.
- AES-256-CBC encryption is applied to the private key before it’s written to the file. The key for this encryption is stored separately and can be protected using a passphrase (see section “Local Data Encryption” for details).

Windows Phone

- The private key is stored in a file in the app’s private data directory.
- Other apps cannot access this file.
- The [Data Protection API \(DPAPI\)](#) is used to encrypt the private key before it’s written to the file. If the user has set a passphrase, it is used as “optional entropy” for the encryption.

Address Book Synchronization

Threema optionally lets the user discover other Threema contacts by synchronizing with the phone's address book. If the user chooses to do so, the following information is uploaded through a TLS connection to the directory server:

- HMAC-SHA256 hash of each email address found in the phone's address book
 - Key: 0x30a5500fed9701fa6defdb610841900febb8e430881f7ad816826264ec09bad7
- HMAC-SHA256 hash of the E.164 normalized form of each phone number found in the phone's address book
 - Key: 0x85adf8226953f3d96cfd5d09bf29555eb955fcd8aa5ec4f9fcd869e258370723

The directory server then compares the list of hashes from the user with the known email/phone hashes of Threema IDs that have been linked with an email address and/or phone number. Any matches are returned by the server as a tuple (Threema ID, hash). Only those hashes that have already been submitted by the user are returned (i.e. if the user submits an email hash which then matches a Threema ID, the server will only return the email hash of that ID and not the linked phone number's hash, even if one exists). After returning the matches to the client, the directory server discards the submitted hashes.

Use of HMAC keys

The reason why HMAC-SHA256 is used instead of plain SHA256 is not for obfuscation, but as a best practice to ensure that hashes generated by Threema are unique and do not match those of any other application. Obviously the keys need to be the same for all users, as random salting (such as is used when hashing passwords for storage) cannot be used here because the hashes of all users must agree so that matching contacts can be found.

A word about hashing phone numbers

Due to the relatively low number of possible phone number combinations, it is theoretically possible to crack hashes of phone numbers by trying all possibilities. This is due to the nature of hashes and phone numbers and cannot be solved differently (using salts like for hashing passwords does not work for this kind of data matching). Therefore, the servers treat phone number hashes with the same care as if they were raw/unhashed phone numbers. Specifically, they never store hashes uploaded during synchronization in persistent storage, but instead discard them as soon as the list of matching IDs has been returned to the client.

Linking

If a user chooses to link their Threema ID to an email address or a phone number, the directory server verifies that the user actually owns the email address or phone number in question.

- For email addresses, a verification email with a hyperlink is sent to the user. The user must open the hyperlink and confirm in the browser before the ID link is established.
- For phone numbers, the directory server sends an SMS message with a random 6 digit code. The user must enter that code in the app, which sends it back to the server to verify the phone number.
 - If the user cannot receive the SMS message, he/she may choose to receive an automated phone call in which the code is read out.
 - Note that on Android devices, the verification SMS message is automatically received and processed by the app in most cases.

ID Revocation

Threema users may revoke their ID at any time by going to a website (<https://myid.threema.ch/revoke>) and entering their Threema ID and a revocation password that they have set beforehand. Revoked IDs can no longer log in, and they will be shown in strikethrough text on (or, depending on the users' settings, disappear from) the contact lists of other users within 24 hours. Messages cannot be sent to revoked IDs.

Revocation passwords are hashed with SHA256 on the client side, but only the first four bytes are sent to and stored/compared on the directory server. This mitigates the possibility of brute-force hash cracking on the server side to recover the user's password. Obviously, the downside is that the stored key has significantly less entropy than a reasonable user password and a large number of possible passwords correspond to each key. However, the server can easily limit the number of key revocation attempts over a period of time.

An Example

A user has set an eight character revocation password with a combination of characters consisting of the lowercase latin alphabet (a-z) and digits. This results in $36^8 \approx 2^{41.36}$ combinations.

Therefore, a given key can correspond with over 600 different passwords of this length and character set. On the other hand, the chances of randomly choosing the correct hash value without knowing anything about the password are $1/2^{32} \approx 0.000000023\%$. Assuming that a user gets three attempts per day, the chances of finding the correct hash value within 10 years is $\approx 0.000255\%$.