Morse.

Morse Messenger Security White Paper

Technical white paper

Version described	Morse 0.2.5
iPhone app	0.2.5
Android app	0.2.5
Morse core	0.1.0
Date	October 2026

Contents

Introduction 3
Terms 3
Account registration 4
Starting a session 5
Receiving a session 6
Exchanging messages 7
Transmitting media and other attachments 9
Group messages 10
Call setup 11
Live location 12
Verifying keys 12
Notifications 13
On the phone 13
What the server stores 14
Transport security 15
What end-to-end encryption means for Morse 15
Conclusion 16

Introduction

This paper explains how Morse Messenger protects messages, files and calls. It covers the keys each phone holds, how two phones set up a secure session, how messages, files, groups and calls are encrypted, and what the server stores.

Morse is built on the Signal Protocol, including its post-quantum key agreement (PQXDH) and its Sparse Post-Quantum Ratchet (SPQR). Morse implements the protocol in its own library, which both the iPhone and the Android app use. Where Signal defines the protocol, that library is tested against Signal's libsignal and produces the same bytes for the same inputs.

Messages, files, voice notes and calls are encrypted on the sender's phone and can only be decrypted on the recipient's phone. The Morse server passes them on but cannot read them. Because the session keys keep changing, the session keys taken from a phone today cannot be used to decrypt previously transmitted messages.

A Morse account lives on one phone. There are no linked devices.

Terms

Public key types

- **Identity key pair.** A long-term Curve25519 key pair, made at registration. The same key is used for Diffie-Hellman agreement and, through XEdDSA, for signatures.
- **Signed prekey.** A medium-term Curve25519 key pair, signed by the identity key. A new one is made every two days.
- **One-time prekeys.** A stock of Curve25519 key pairs, each used for at most one session setup. The phone refills the stock as needed.
- **Post-quantum prekeys.** ML-KEM-1024 key pairs, signed by the identity key. There is one reusable key, replaced every two days, and a stock of one-time keys that the phone refills as needed.

Session key types

- **Root key.** A 32-byte value used to create chain keys.
- **Chain key.** A 32-byte value used to create message keys.
- **Message key.** An 80-byte value used to encrypt one message: a 32-byte AES-256 key, a 32-byte HMAC-SHA256 key and a 16-byte IV.

- **SPQR key.** A 32-byte value that the post-quantum ratchet (SPQR) gives every message. The ratchet starts from the SPQR starting key and advances through ML-KEM-768 exchanges (see “The post-quantum ratchet”).

Other key types

- **Sender key.** A chain key and a signing key that a member uses for the messages it sends to a group.
- **Call key.** A Curve25519 key pair whose public half others use to seal call setup messages to this phone.
- **Call-signing key.** A Curve25519 key pair that signs call setup messages. Because the identity key is not available while the app is locked, the identity key certifies this key for 30 days at a time, so a locked phone can still answer a call.
- **Preview key.** A Curve25519 key pair that notification previews are sealed to (see “Notifications”).

Account registration

A Morse account is not tied to a phone number, an email address or a real name. The phone picks a random Morse ID (8 hexadecimal characters from 4 random bytes) and the server picks a random inbox ID (16 random bytes) that messages are delivered to.

At registration the phone sends the server its Morse ID, its public identity key, its signed prekey with its signature, a batch of one-time prekeys and its reusable post-quantum prekey with its signature. The server checks both signatures against the identity key before it accepts the account. It then stores these public keys with the Morse ID and the inbox ID. Private keys never leave the phone.

The phone also makes a 32-byte deletion token, which can delete the account later. The server stores only its SHA-256 hash.

To log in, the phone asks the server for a challenge of 32 random bytes and signs the label `"MorseLogin-v1"` followed by the challenge with its identity key. The server checks the signature against the stored identity key and issues a login token. The token is 32 random bytes; the server stores only its SHA-256 hash, and the token expires 14 days after it was last used.

After logging in, the phone uploads its one-time post-quantum prekeys, its call key and its call-signing key with the identity key’s certificate. The server checks the signature on every post-quantum prekey.

Starting a session

Before two phones exchange messages, the sender sets up a session with the recipient. Once the session exists, it is used until one of the phones loses it, for example after a reinstall.

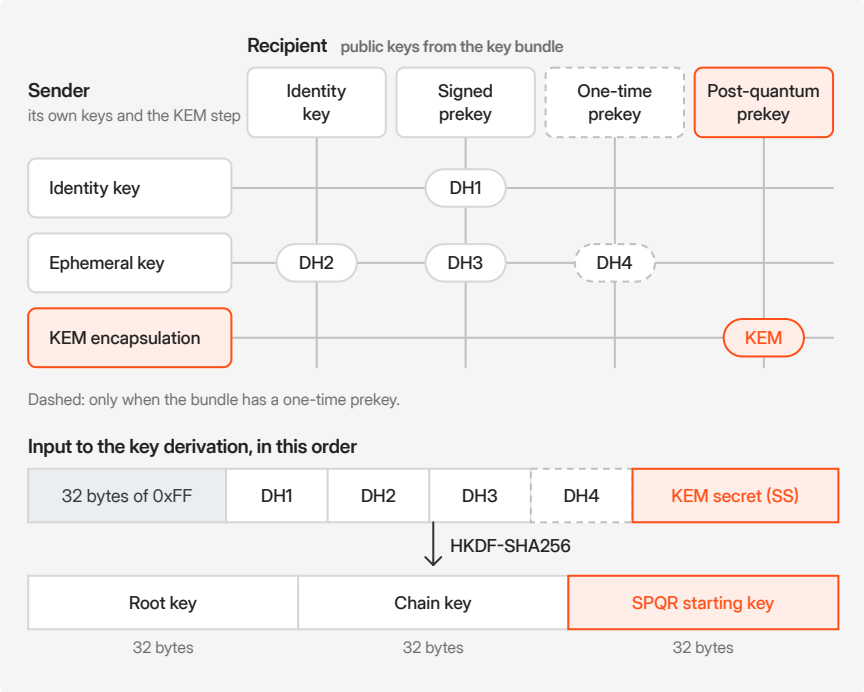


Figure 1 The key agreement (PQXDH). Each DH result combines one key of the sender with one public key of the recipient; the KEM secret comes from encapsulating to the post-quantum prekey. All results go into one key derivation.

1. The sender asks the server for the recipient's key bundle: the identity key, the signed prekey with its signature, a single one-time prekey if any are left and a single post-quantum prekey with its signature. The server hands out each one-time prekey only once. While the recipient has one-time post-quantum prekeys left, the bundle carries one of those; otherwise it carries the reusable post-quantum prekey.
2. The sender checks the two signatures with the recipient's identity key. If either fails, it stops.
3. The sender saves the recipient's identity key as $I_{recipient}$, the signed prekey as $S_{recipient}$, the one-time prekey as $O_{recipient}$ and the post-quantum prekey as $K_{recipient}$.
4. The sender generates an ephemeral Curve25519 key pair, E_{sender}, and loads its own identity key as I_{sender}.
5. The sender runs ML-KEM-1024 encapsulation against $K_{recipient}$, which gives a 32-byte shared secret SS and a ciphertext CT.

6. The sender calculates the master secret (the four ECDH values are DH1 to DH4 in Figure 1):

   ```
   master_secret = 0xFF (32 bytes)
                || ECDH(I_sender, S_recipient)
                || ECDH(E_sender, I_recipient)
                || ECDH(E_sender, S_recipient)
                || ECDH(E_sender, O_recipient)
                || SS
   ```

   If no one-time prekey was available, the fourth ECDH is left out.

7. The sender uses HKDF-SHA256 with the info `"WhisperText_X25519_SHA-256_CRYSTALS-KYBER-1024"` to turn the master secret into a root key, a chain key and the SPQR starting key. The label keeps the algorithm's earlier name, Kyber.

To make it hard to use up a person's one-time prekeys, the server limits how often key bundles can be fetched.

Receiving a session

The sender can send messages right away, even when the recipient is offline. Until the recipient answers, every message the sender sends carries what the recipient needs to build the same session: E_{sender}, I_{sender}, the IDs of the prekeys used and the ML-KEM ciphertext CT.

When the recipient receives a message with this information:

1. It checks that E_{sender} is a canonically encoded Curve25519 public key in the prime-order subgroup.
2. It looks up its own signed prekey, one-time prekey and post-quantum prekey by the IDs in the message, and runs ML-KEM-1024 decapsulation on CT to obtain SS.
3. It calculates the same master secret with its private keys and derives the same root key, chain key and SPQR starting key.
4. After the message has been decrypted, it deletes the one-time prekey and the one-time post-quantum prekey that were used, if any.

The recipient also checks that the session was built with the identity key it stored for this contact. A message from a session with any other identity key is refused.

Exchanging messages

Once a session exists, messages are encrypted with AES-256 in CBC mode. Each message carries an authentication code, HMAC-SHA256 truncated to 8 bytes, which also covers the identity keys of both phones.

Every message uses a new message key, and a used message key cannot be calculated again from the session state.

Calculating a message key from a chain key

1. `seed = HMAC-SHA256(chain_key, 0x01)`
2. `message_key = HKDF-SHA256(salt = spqr_key, input = seed, info = "WhisperMessageKeys")`, 80 bytes, where `spqr_key` is this message's SPQR key.
3. `chain_key = HMAC-SHA256(chain_key, 0x02)`, which moves the chain key forward.

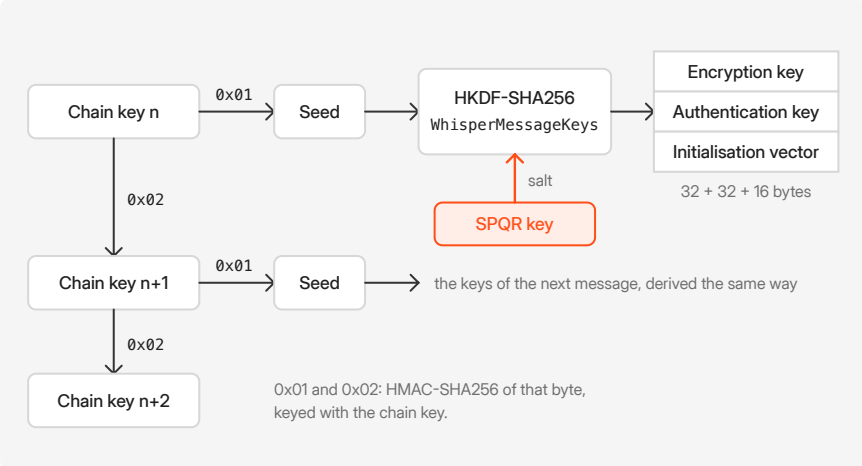


Figure 2 From chain key to message keys. Each step down the chain gives the keys of one message.

Because message keys and the next chain key are derived from the chain key with one-way functions, a message key cannot be used to derive any chain key, earlier or later.

Calculating a chain key from the root key

Every message carries the sender's Curve25519 ratchet public key. When a reply with a new ratchet key arrives, the phone combines the two current ratchet keys, R_{sender} and $R_{recipient}$:

1. `ephemeral_secret = ECDH(`R_{sender}`,` $R_{recipient}$`)`

2. `root_key, chain_key = HKDF-SHA256(salt = root_key, input = ephemeral_secret, info = "WhisperRatchet")`

A chain is only ever used by one sender, so message keys are never reused. Messages can arrive late, out of order or not at all without breaking the session.

The post-quantum ratchet

Alongside this ratchet, every session runs SPQR, Signal's post-quantum ratchet; together with the Double Ratchet described above, Signal calls this the Triple Ratchet. The two phones keep exchanging ML-KEM-768 keys and ciphertexts. These are split into erasure-coded chunks that are carried in ordinary messages, so a lost message does not stop the exchange. Each completed exchange adds a new secret to the SPQR chain, and the chain gives every message its SPQR key. That key is the salt when the message key is derived (step 2 above). An attacker therefore has to break both the elliptic-curve ratchet and the post-quantum ratchet to read a message sent in the session.

The sealed envelope

Every one-to-one message travels in a sealed envelope addressed to the recipient's inbox. Inside it are the sender's Morse ID and identity key and the encrypted message. The envelope is encrypted to the recipient's identity key with a fresh Curve25519 key pair for every message: an AES-256 key is derived with HKDF-SHA256 and the envelope is encrypted with AES-256-GCM. The server stores a waiting message with the recipient's inbox and no sender, and deletes it once the recipient's phone confirms it has stored it, or after 24 hours.

The envelope itself does not authenticate the sender. The message inside does: its authentication code covers both identity keys, and the recipient checks that the session was built with the identity key it holds for that contact.

Padding

Before a message is encrypted, the phone pads it to a length one byte short of a multiple of 80 bytes. CBC mode's PKCS#7 padding then adds the last byte, so every ciphertext is a whole multiple of 80 bytes and its length says little about what is inside. Group messages are padded the same way.

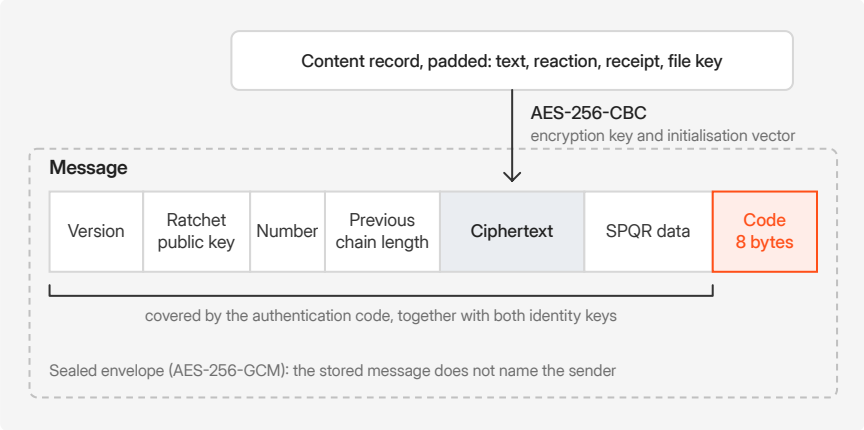


Figure 3 The layers around one message, from the content outward.

Transmitting media and other attachments

Photos, videos, voice notes and files are end-to-end encrypted:

1. The sender's phone generates a random 32-byte AES-256 key for the file.
2. It encrypts the file with AES-256-GCM in frames of 64 KiB. Each frame has its own random IV, and its authenticated data includes its position and whether it is the last frame, so frames cannot be reordered, cut off or added.
3. It uploads the encrypted file to the server under a random file ID.
4. It sends a normal encrypted message to the recipient with the file key, the file ID and the file's type, size and preview image.
5. The recipient's phone downloads the encrypted file and decrypts it frame by frame. It rejects the file if any frame fails authentication or the header does not match the frames. It then erases the file key.

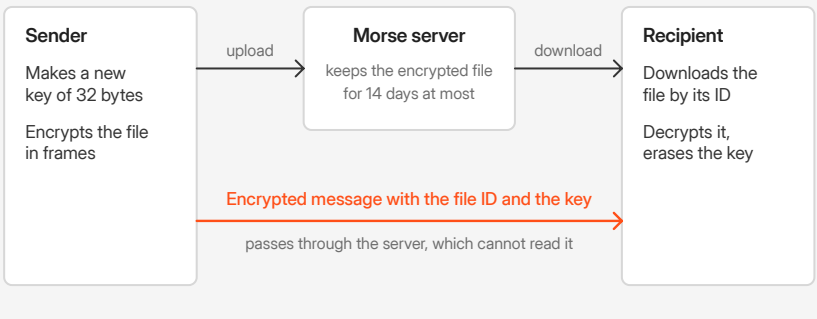


Figure 4 Two paths. The encrypted file goes to the server. Its key travels inside an encrypted message.

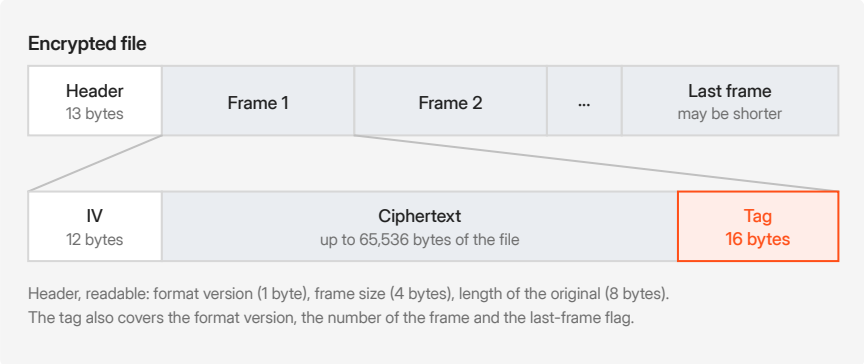


Figure 5 An encrypted file: a readable header, then one frame for every 64 KiB of the original.

Encrypted files stay on the server for at most 14 days. When someone deletes their own message for everyone, their phone also deletes the message's files from the server.

Before a photo is sent, the phone writes it again as a new image, which removes its location, date and camera details. Videos are rebuilt from their picture and sound only, which removes the location and date stored in the video file. Files and documents are sent exactly as they are, with their file name.

Group messages

Group messages use the pairwise sessions described above to hand out sender keys, as in the Signal Protocol.

The first time a member sends a message to a group:

1. The sender generates a random 32-byte chain key and a Curve25519 signing key pair.
2. It puts the chain key and the public signing key into a sender key message.
3. It sends the sender key message to each member through their pairwise session.

Then, for every message to the group:

1. The sender derives a message key from the chain key (HMAC-SHA256 and HKDF-SHA256 with the info `"WhisperGroup"`) and moves the chain key forward.
2. It encrypts the message once, with AES-256 in CBC mode.
3. It signs the encrypted message with its signing key.

4. It sends the one encrypted message to the server together with the list of recipients and, for each of them, a sealed envelope that tells only that recipient who sent it. The server delivers the message to each listed recipient.

Whenever a member leaves or is removed, every remaining member's phone discards its sender key and makes a new one for its next message to the group, which it hands only to the current members. Someone who has left cannot read new messages.

Encrypted groups

Morse stores groups on the server using the Signal Private Group System, in encrypted form. Each group has a master key that is shared with its members and never with the server. From it, the phones derive the keys that encrypt the group's data:

- Every member is stored only as an encrypted entry that the server cannot read.
- The group's name, description, picture and disappearing-message timer are encrypted with AES-256-GCM-SIV.

To read or change a group, a member presents an anonymous credential with a zero-knowledge proof that a given encrypted entry belongs to it. The server checks that this entry is in the member list without learning who the member is. A message to a group carries a token that proves the sender may send to exactly the listed recipients.

The server can see each member's role and the number of members, which it needs to apply the group's rules.

Call setup

Voice and video calls are end-to-end encrypted. Only contacts, and people you sent a contact request to, can ring you.

1. The two phones set up the call by exchanging call setup messages: the caller's offer, the callee's answer, and the relay addresses through which each side can be reached.
2. Every call setup message is signed with the sender's call-signing key over the label `"MorseCallSignal-v1"` and the message, and is sent with the identity key's certificate for that call-signing key.
3. The signed message is sealed to the other phone's call key with a fresh Curve25519 key pair, HKDF-SHA256 and AES-256-GCM.

4. The receiving phone accepts it only if the certificate is valid under the identity key it holds for that contact, the signature is valid and the certificate has not expired. The signed message also names who it is for, so it cannot be passed on to someone else.

The offer and the answer carry the fingerprints of the keys the two phones then use to agree on the call's encryption (DTLS-SRTP, as in WebRTC). Because those fingerprints arrive inside signed messages, a third party cannot insert itself into the call. The audio and video are encrypted with SRTP between the two phones.

Every call goes through a relay server, so the two phones never learn each other's network address. The relay only forwards encrypted audio and video. The credentials a phone uses at the relay are random and say nothing about who is calling.

Live location

A shared location and every live update are ordinary end-to-end encrypted messages, sent through the same pairwise session or group sender key as other messages.

Verifying keys

Morse users can check that nobody, not even Morse, is standing between them and a contact. Each contact has a Safety Number page with a 60-digit number and a QR code.

To calculate the 30 digits for one side:

1. Hash the side's identity key and inbox ID together with SHA-512, then hash the result with the key again, 5,200 times in total.
2. Take the first 30 bytes and split them into six 5-byte pieces.
3. Turn each piece into 5 digits by reading it as a big-endian number and taking the remainder after dividing by 100,000.

The 60-digit number is the two sides' 30 digits, the lower one first, so both phones show the same number. The QR code holds the first 32 bytes of each side's hash. When one phone scans the other's code, the two must match exactly.

Morse also compares keys with the server. When the Safety Number page opens, the phone asks the server for this contact's current identity key and compares it with the key it saved when the contact was added. If they match,

the contact's page shows "Encryption checked"; if they do not, the chat shows a warning. This check only confirms that the server still reports the key the phone saved. Only comparing the number in person or scanning the code proves that nobody was in between; after that, the user can mark the contact as verified.

Notifications

A push notification never carries a readable message. It wakes the phone, which then fetches the encrypted message from the server over its own connection. The readable text in a message push is always the same generic line, "New message", so Apple's and Google's notification services never see who wrote what.

When the app has no lock, the phone decrypts the message and builds the notification from it.

When the app is locked with a PIN or biometrics, the phone cannot open its database or its sessions in the background. For that case the sender's phone sends a short sealed preview with each message, which travels with the push: the sender's name and the start of the message. It is signed with the sender's identity key and sealed to the recipient's preview key. The sender only learns that key from the recipient's own messages, never from the server. A preview is sealed outside the session, with Curve25519 only, so it does not have the session's forward secrecy or post-quantum protection. The locked phone shows a preview only if it opens with its preview key, carries a valid signature by the identity key the phone holds for that contact, names this account as its recipient, is recent, has not been shown before and comes from a contact who is not blocked. Anything else shows just "New message".

Turning previews off replaces the preview key, so previews sealed to the old key can no longer be opened.

On the phone

The encrypted database

Messages, contacts and groups are stored in an encrypted database (SQLCipher) with a random 32-byte key. The account's keys are stored separately. On the iPhone, the keys are Keychain items available only on this phone. On Android, they are protected by keys in the Android Keystore.

The app lock

Users can lock the app with a PIN or with biometrics. With a PIN, the database key and the account keys are encrypted twice: with AES-256-GCM under a key derived from the PIN, and again under a key that cannot be taken out of the phone (held by the Secure Enclave on the iPhone and the Android Keystore on Android). The PIN itself is never stored, and repeated wrong PINs make the user wait longer each time.

With a lock set, the app locks as soon as it leaves the screen. Locking closes the database, and the app overwrites its copies of the database key and the account keys. To ring for calls and show previews while locked, the phone keeps what it needs for that outside the PIN protection, such as the call keys, the preview key and the public identity keys of contacts. None of these opens the database or a session.

The panic button

Users can turn on a panic button that erases the account in one step, without unlocking the app. It can be shown in the app and on the app's unlock screen. When pressed, the phone asks the server to delete the account with the deletion token, then erases Morse's keys, database and files on the phone, whether or not the server answered. Users can choose to also erase the photos and videos Morse saved to the phone's gallery.

Backups

On Android, Morse's data is excluded from cloud backups and from transfers to a new phone. On the iPhone, received files are excluded from iCloud and computer backups, and the keys that open Morse's data are stored for this phone only, so a backup cannot be read on any other device.

Disappearing messages

A chat can delete its messages automatically after 30 seconds, 5 minutes, 1 hour, 8 hours, 1 day, 1 week, 4 weeks or 90 days. New chats use 90 days unless the user changes this. The timer starts when the message is sent, and every phone in the chat deletes the message and its files when it runs out.

What the server stores

The server stores:

- The account's Morse ID, inbox ID, public keys and avatar colour.
- The account's subscription status.
- Login and deletion tokens, only as SHA-256 hashes.

- The push token for Apple's or Google's notification service.
- The encrypted profile (name and picture), which the server cannot read.
- Waiting messages, encrypted and without a sender, until they are delivered or for at most 24 hours.
- Encrypted files, for at most 14 days.
- Groups, with encrypted members and encrypted details.
- Reports: the reported account or group, the account that reported it, the reason and, only with the reason Other, the reporter's own words. A report contains no messages, photos or files. It is deleted once it is handled, and after at most 30 days unless the law requires keeping it longer.

The server has no phone numbers, email addresses, names, contact lists or block lists. Contacts are kept only on the phone. Morse does not read the phone's address book.

When an account is deleted, the server deletes its keys, tokens, profile, waiting messages and push token.

Transport security

All traffic between the app and the server uses TLS 1.2 or newer: HTTPS for requests and a secure WebSocket for live delivery. Messages and files inside this connection are already end-to-end encrypted.

Users have no password. A phone logs in by signing a challenge with its identity key, and for logging in and deleting the account the server keeps only public keys and SHA-256 hashes of the login and deletion tokens. If the server's user database were ever exposed, it would reveal no credential that lets someone log in as a user.

What end-to-end encryption means for Morse

Morse defines end-to-end encryption as communication that stays encrypted from the sender's phone to the phones it is meant for, where no third party can read the content in between, not even Morse. A third party here is anyone who is not the sender or a recipient in the conversation: the network, the call relay, Apple's and Google's notification services, and Morse itself.

In Morse this covers messages, files, voice notes, calls, group details, profiles and notification previews. Nobody else can read these messages, and a report does not change that: it sends only who is reported and why, never messages.

Conclusion

Every Morse message and file is protected with the Signal Protocol. The one-to-one sessions, which also carry the group sender keys, use its post-quantum key agreement (PQXDH) and its post-quantum ratchet (SPQR). Calls are set up with signed, sealed messages and encrypted with SRTP between the two phones. The server keeps no phone numbers, no contact lists, no readable group member lists and no messages after they are delivered.