Crypho Security Whitepaper

Crypho AS

Crypho is an end-to-end encrypted enterprise messenger and file-sharing application. It achieves strong privacy and security using well-known, battle-tested encryption methods. This whitepaper describes key concepts and algorithms used in Crypho.

1 Overview

Crypho uses several layers of protection to provide privacy and security. All data is end-to-end encrypted without Crypho or anyone else having access to the encryption keys. There is no way for Crypho or a third party to gain access to any of the data. Even if the data is intercepted, stolen or seized, it is worthless without access to the members' keys.

2 Generating and storing user encryption keys

Every member of Crypho has the following keys that are used to protect her communications:

- A 256-bit key, K_s
- A public/private El-Gamal key-pair $P_{encryption,pub}/P_{encryption,pr}$
- A public/private ECDSA key-pair $P_{signing,pub}/P_{signing,pr}$

When a member registers with Crypho, she creates a passphrase. This passphrase is never communicated to the server or stored on the member's device. The passphrase is further strengthened using the Scrypt[1] key-derivation algorithm to generate a 512-bit long key K which is then split into two such that:

$$\begin{aligned} K &= scrypt(passphrase, salt), \\ K &= K_s || K_p \end{aligned} \tag{1}$$

where *salt* is derived from the user's email address.

The client also automatically creates an Elliptic-Curve-Cryptography (ECC) El-Gamal key-pair as well as an Elliptic-Curve-Digital-Signature-Algorithm (ECDSA) key-pair. The private parts ($P_{encryption,pr}$, $P_{signing_pr}$) of the key-pairs are then encrypted with AES-256, using keys derived from K_s thus giving the user encrypted versions that only she can decrypt:

$$\begin{aligned} k_{encryption} &= HMAC(K_s, S_{encryption}), \\ k_{signing} &= HMAC(K_s, S_{signing}), \\ P'_{encryption,pr} &= AES(P_{encryption,pr}, k_{encryption}) \\ P'_{signing,pr} &= AES(P_{signing,pr}, k_{signing}) \end{aligned} \tag{2}$$

The public keys $P_{encryption,pub}$, $P_{signing,pub}$ and the encrypted $P'_{encryption,pr}$, $P'_{signing,pr}$ are sent to the server and stored. The server never gets to know the passphrase of the user or has any way of deriving it from what it knows. Only the member (using her passphrase) can re-generate K_s and decrypt her $P'_{encryption,pr}$, $P'_{signing,pr}$.

The second part of K, K_p is hashed and stored on the server. It is used for one-step two-factor authentication see section 3

3 Authentication

Crypho uses the Time-based One-time Password Algorithm(TOTP)[2] for two-factor authentication. Upon registration the server generates a TOTP secret. The member can then choose to use SMS/text messages to receive TOTP tokens, or to generate them through Crypho's mobile app.

To authenticate, the member needs to enter her passphrase, TOTP token and email address. From the passphrase and using Eq. 1 the user obtains K_p which is hashed again and sent to the server together with the TOTP token. Hence, the server can verify that the user knows both the passphrase and a valid token and return $P_{encryption,pub}$, $P'_{encryption,pr}$, $P_{signing,pub}$, $P'_{signing,pr}$ to the user. The user having computed K_s can then use Eq. 2 to decrypt her private keys.

Crypho limits the number of attempts and will notify the user and staff in case of multiple failed attempts.

4 Instant messaging & file end-to-end encryption

When a member initiates a conversation (by accepting an invitation or by creating a group conversation) she generates a random 256bit long conversation-specific key P_c. This key is then further encrypted with the public keys of all conversation participants $P_{encryption,pub}$ and stored on the server. Thus, only authenticated users who have access to their private key $P_{encryption,pr}$ can successfully decrypt P_c and consecutively decrypt the messages and files of the conversation.

In addition P_c is hashed and signed by its creator using her $P_{signing,pr}$. This allows the rest of the members of the conversation to verify its authenticity.

In case a user is removed from a conversation, a new key is automatically generated by the user who removed the member and shared through the remaining members' public keys. In case a user leaves the conversation, resets her passphrase, or deletes her account, the conversation is "tainted", and the next time another member is online she will create a new key and share it.

5 User verification and signing

Each user has a "fingerprint" F_u calculated as

$$F_u = SHA256(P_{encryption,pub}||P_{signing,pub}), \tag{3}$$

of which the first 64 bits are used. Users can sign each other's fingerprint in an out-of-band fashion verifying thus that the person behind the keys is who they claim they are. To facilitate communication of the keys, Crypho encodes the fingerprint as a QRCode that can be scanned by mobile apps as well as a sequence of words using the algorithm described in *mnemonic.js*[3].

For example, Alice has the fingerprint `1fa635e9a299cc6e` in hexadecimal format. The fingerprint encodes to the sequence `[rebel, praise, flirt, truck, park, yet]`, which Alice can communicate to Bob in a real-life meeting or over the phone. Bob can then input the sequence in Crypho, verify Alice's public keys and sign the fingerprint with his own $P_{signing,pr}$. Later he can always be assured that a key signed by Alice in one of his conversations has been indeed generated by Alice.

6 Web app security

Regardless of the encryption mechanisms described above, all communications between the Crypho servers and web browsers or mobile applications are encrypted using HTTPS/TLS (Transport Layer Security). While this makes no difference to the privacy of the data transmitted (since they are end-to-end encrypted) it helps further safeguarding client-server communications and mitigate man-in-the-middle attacks.

Crypho pays close attention to security announcements on new vulnerabilities on the HTTPS/TLS protocols and adapts if necessary in a timely fashion.

To protect users from cross-site scripting attacks (XSS), Crypho's web app uses Content Security Policies to declare approved sources of content that are allowed to run in the Crypho web application.

7 Mobile app security

User interactions such as entering a long passphrase can be a challenge on mobile devices given their small form factor. To solve this problem Crypho developed a secure storage mechanism that uses native device security when available and complements it with strong cryptography when necessary. Thus we allow the member to authenticate without entering her passphrase every time, by cryptographically storing her key-pairs on the device. The member can opt-out of this feature, in which case her passphrase will be required every time she uses the app. This is recommended in hostile environments.

To enhance security and transparency, Crypho has released relevant libraries as open-source software on Github[4, 5].

8 Privacy, anonymity and metadata

Crypho protects the contents of your conversations and files you share by end-to-end encrypting them with keys that are only under your control.

Crypho is not anonymous. It aims to keep your data private, while providing all of the features expected by a communication service.

For example, Crypho's server is aware of when a new message is posted in a conversation and notifies users via the web or mobile notification system. Since Crypho does not have access to the contents of the conversations, it is impossible to include the content of the message in the notifications (which would in itself pose a security risk).

Our servers have no knowledge of your message contents, but store things like your email address, your telephone number, your contacts in Crypho, the IP addresses you use. Crypho also stores additional data such as the time you logged-in, or when a message was sent and by whom. Crypho does not disclose any of this information or share it with third parties, except when needed in order to fulfill its operations, for example sending you an SMS/text message.

9 Operational security/hosting

Crypho maintains strict policies on operational security. All our code is peer--reviewed for security before deployment and we maintain a large number of automatic tests that target security specifically. No cloud services are used for hosting or storing of the user data. Crypho is hosted on dedicated servers in Norway in order to keep our infrastructure under strict control. All your data is protected by Norwegian law.

10 Responsible vulnerability disclosure

We perform regular security audits internally and communicate regularly with security experts globally. Our engineering team has strong security-related background and experience. However, no software is without bugs. If a security vulnerability is found, Crypho will alert users immediately and disclose all information relating to the vulnerability after it is fixed. We strive for transparency and trust.

11 Data ownership

All information exchanged using Crypho's service is owned by the members participating in the conversation. Crypho does not not claim any ownership of this data. Our architecture additionally makes it impossible for us to decrypt and access it.

12 Cryptography details

Crypho uses well known and battle-proven encryption algorithms and libraries to ensure privacy as shown in Table 1.

ECC Encryption	The ElGamal Elliptic Curve 384bit prime curve is used.
AES	Authenticated AES with 256bit keys in CCM mode.
Scrypt	Scrypt algorithm with $N = 16384, r = 8, p = 1$.
Randomness	All random numbers/IVs/keys are generated using secure PRNGS. On browsers and desktop apps Crypho uses `crypto.getRandomValues()`, whereas on mobile devices the native secure PRNGs are used instead [4]
Crypto primitives	All crypto primitives used by Crypho are based on the Stanford Javascript Crypto Library [6].

Table 1: *Cryptography primitives*

Additionally the following Crypho open-source libraries are used on mobile for native fast cryptographic operations:

React native secure random library for iOS & Android `https://github.com/Crypho/react-native-secure-random`

React native scrypt libary for iOS & Android `https://github.com/Crypho/react-native-scrypt`

References

[1] The scrypt key derivation function http://www.tarsnap.com/scrypt.html

[2] Time-based One-time Password Algorithm https://en.wikipedia.org/wiki/Time-based_One-time_Password_Algorithm

[3] mnemonic.js https://github.com/ggozad/mnemonic.js

[4] Secure random PRNG library for iOS & Android https://github.com/Crypho/react-native-secure-random

[5] Scrypt plugin for iOS & Android https://github.com/Crypho/react-native-scrypt

[6] Stanford Javascript Crypto Library https://github.com/bitwiseshiftleft/sjcl