SAFESWISS®

SECURE COMMUNICATION

ENCRYPTION WHITEPAPER

SAFESWISS ENCRYPTION WHITEPAPER 2021

SafeSwiss® uses modern cryptography based on open-source components that strike an optimal balance between security, performance, message size and delivery. In this whitepaper, the algorithms, and the technical design behind the encryption in SafeSwiss® are explained.

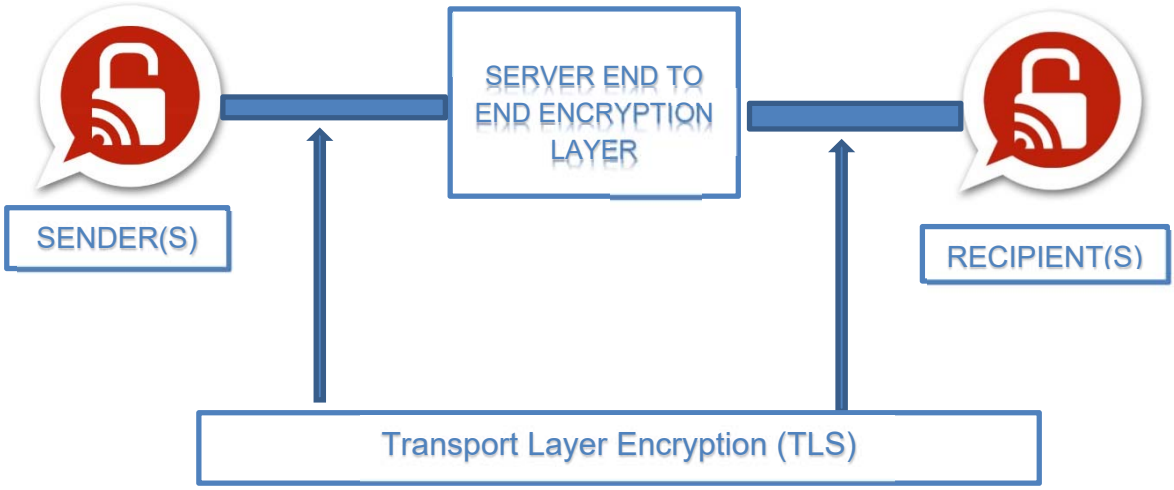
CONTENTS

SAFESWISS® OVERVIEW

SafeSwiss® 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 SafeSwiss 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.



SafeSwiss® Message Delivery Receipts

When a SafeSwiss® user sends a message:

- One black Tick check — message delivered to the SafeSwiss server
- Two black Tick checks — message has been delivered to recipient
- Two Red Tick checks — message has been read (Recipient has opened SafeSwiss and opened the conversation with the message) and the message has been deleted from SafeSwiss® servers

.

END-TO-END ENCRYPTION

In SafeSwiss® all messages (simple SMS text messages, or contacts media files such as images, videos, or audio recordings) are always end-to-end encrypted. For this purpose, each SafeSwiss® user has a unique asymmetric key pair consisting of a public and a private key based on AES-256 Elliptic Curve Cryptography. These two keys are mathematically related, but it is technically feasible to calculate the private key given only the public key.

Key Generation and Registration

When a SafeSwiss® 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 random storing it securely on the user's device and calculating the corresponding public key over the Elliptic Curve (Curve25519).
2. The SafeSwiss® app sends the public key to the SafeSwiss® server(s)
3. The SafeSwiss® server stores the public key and assigns a new random SafeSwiss® ID, consisting of 8 characters out of A-Z/0-9.
4. The SafeSwiss® app stores the received SafeSwiss® ID along with the public and private key in secure storage on the user's device.

Key Distribution and Trust

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

- A full 8-character SafeSwiss® ID
- A hash of a E.164 mobile phone number that is linked with a SafeSwiss® ID
- See section "Address Book Synchronization" for details on the hashing

Key Fingerprints

The SafeSwiss® app displays a key fingerprint for each contact, and for the user's own identity. This can be used to compare public keys manually. A key fingerprint is created by hashing the raw public key with SHA-256 and taking the first 16 bytes of the resulting hash as the fingerprint.

MESSAGE ENCRYPTION

SafeSwiss® uses the "Box" model of the NaCl Networking and Cryptography Library https://nacl.cr.yp.to/ 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

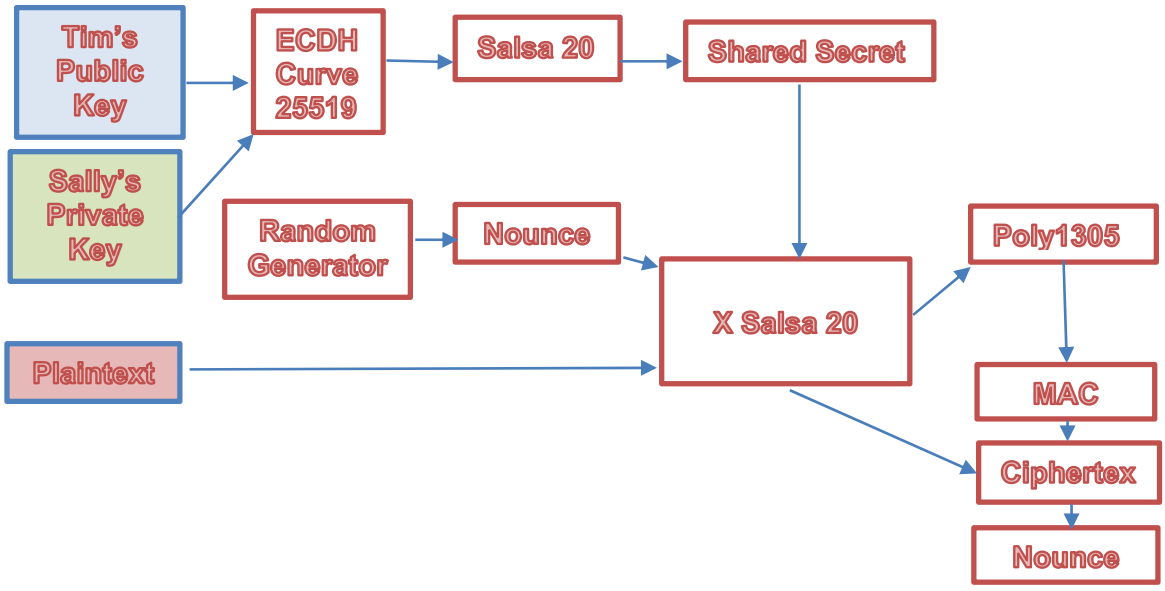
1. Sally and Tim have each generated a key pair consisting of a public and a private key.
2. Sally has obtained the public key from Tim over an authenticated channel.

Procedure to encrypt a message

1. Sally 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 Tim'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 Tim's private key and Sally's public key are used instead).
2. Sally generates a random nonce.
3. Sally uses the XSalsa20 stream cipher with the shared secret as the key and the random nonce to encrypt the plaintext.
4. Sally uses Poly1305 to compute a Message Authentication Code (MAC) and prepends it to the cipher- text. A portion of the key stream from XSalsa20 is used to form the MAC key.
5. Sally sends the MAC, Ciphertext and nonce to Bob.

By reversing the above steps and using his own private key and the Sally's public key, Tim can decrypt the message and verify its authenticity.



Group Messaging

SafeSwiss®, groups are managed without any involvement of the servers. SafeSwiss® servers do not know which groups exist and which SafeSwiss® users are members of which SafeSwiss® groups. When a SafeSwiss® user sends a message to a SafeSwiss® group, it is individually encrypted and sent to each other group member. This can 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 applicable SafeSwiss® group.

SafeSwiss® users can back up their private key to Google Drive etc. so that they are able to move their SafeSwiss® ID to another device, or to restore it in case of loss/damage to the device. The SafeSwiss® app automatically reminds the user to do so, as without a backup, there is absolutely no way to recover a lost SafeSwiss® private key.

To generate a backup, the user must first choose a password (min. 8 characters). The

backup data is then generated as follows:

1. Calculate the SHA-256 hash of the following binary string:

   <identity><private key>

   where <identity> is the 8-character SafeSwiss® ID, and <private key> is the 32-byte private key.
   Keep only the first two bytes of the resulting hash. It is used during restoration to verify with reasonable confidence that the provided password was correct.
2. Choose a random 64-bit salt.
3. Derive a 256-bit encryption key from the given password and the random salt using PBKDF2 with HMAC-SHA256 and 100000 iterations:

   key_{enc} = PBKDF2(HMAC-SHA256, password, salt, 100000, 32)
4. Use the XSalsa20 stream cipher with key_{enc} and an all-zero nonce to encrypt a binary string of the following format:

   <identity><private key><hash>

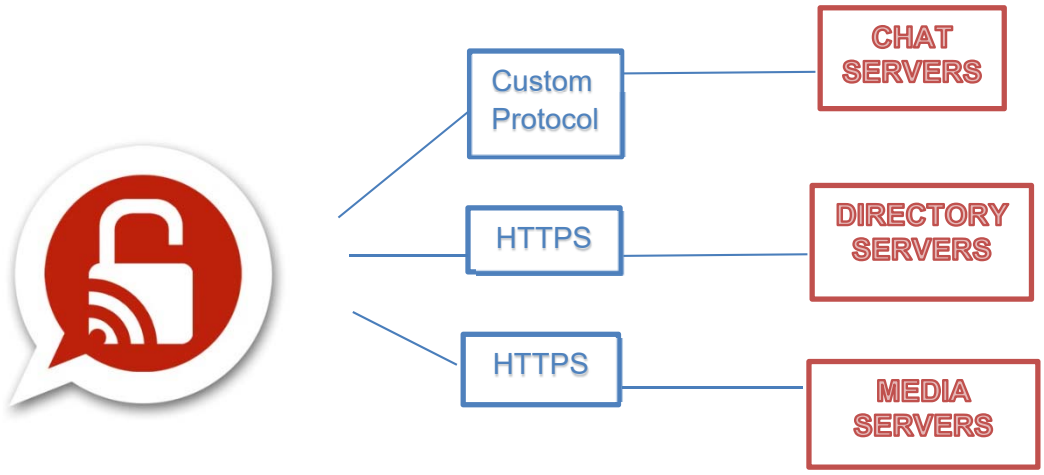
   where <hash> is the two-byte truncated hash calculated in step 1. Note that this can be done using the crypto stream function of NaCl.
5. Prepend the salt to the encrypted string.
6. Base32 encode the result.
7. Split the Base32 encoded string into groups of four characters and separate the groups with dashes. The result will look like this:

   XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX- XXXX-XXXX-XXXX-XXXX-
   XXXX-XXXX-XXXX-XXXX-XXXX-XXXX

   The encrypted backup data consists of 80 characters (A-Z, 2-7). It is also displayed in QR code form so that the user can more easily transfer it to another device.

Client-Server Protocol Description

SafeSwiss® communicates with three different types of servers. To access the directory and to download/upload encrypted media files, HTTPS is used. To transport the actual chat messages, a SafeSwiss® custom protocol built on TCP is used.



SafeSwiss® Chat Protocol TLS (Message Transport Layer)

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

- Provides Perfect Forward Secrecy
  - New temporary keys are always generated whenever the SafeSwiss® 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 roundtrips 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 SafeSwiss® ID

SafeSwiss® Directory Access Protocol

The directory access protocol is used for the following purposes:

- Creating new SafeSwiss® 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 SafeSwiss® 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 SafeSwiss® 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 SafeSwiss® 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.

SafeSwiss® 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.

a. The sender uploads the encrypted media file to a media server via HTTPS.

b. The media server assigns a unique ID for this upload and returns it to the sender.

c. The sender sends an end-to-end encrypted message to the recipient, which contains the media ID and the symmetric key.

d. The recipient receives and decrypts the end-to-end encrypted message, obtaining the media ID and the symmetric key.

e. The recipient uses the media ID to download the encrypted media file from the media server.

f. The recipient decrypts the media file using the symmetric key.

g. The media server automatically deletes the file upon successful download.

CRYPTOGRAPHY DETAILS

SafeSwiss® 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 SafeSwiss® 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

ECC curve used by NaCl (and used by SafeSwiss®) is **not one of the NIST- recommended curves** that have been suspected of containing deliberately selected weakening constants in their specification and design.

For more details, see https://www.nist.gov/system/files/documents/2017/05/09/VCAT-Report-on-NIST-Cryptographic-Standards-and-Guidelines-Process.pdf

The following implementations of the cryptographic algorithms are used:

	Curve25519	**XSalsa20**	**Poly1305**
iOS	*Donna* C implementation	reference C implementation	*"53"* C implementation
Android	reference C implementation (native code)		

Key Lengths

The asymmetric keys used in SafeSwiss® 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.

According to NIST Special Publication 800-57https://csrc.nist.gov/publications/detail/sp/800-57-part- 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.

AES 256 is virtually impenetrable using brute-force methods. While a 56-bit DES key can be cracked in less than a day, AES would take billions of years to break using current computing technology. Hackers would be foolish to even attempt this type of attack.

Random Number Generation

SafeSwiss® 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, SafeSwiss® 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

SafeSwiss® does not use the flawed Dual EC DRBG random number generator. This Number generator was shown to provide backdoor access.

https://arstechnica.com/information-technology/2015/01/nsa-official-support-of-backdoored-dual_ec_drbg-was-regrettable/

https://www.schneier.com/blog/archives/2007/11/the_strange_sto.html

Perfect 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.

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

ephemeral keys. Due to these and the following considerations, SafeSwiss® 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 be- tween the server and the recipient, is orders of magnitude greater than the risk of eavesdropping on the server itself.

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

Reputability
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 reputability (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, SafeSwiss® adds a random amount of PKCS#7 padding to each message before end-to-end encryption.

LOCAL DATA ENCRYPTION

The SafeSwiss® 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

SafeSwiss® 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 SafeSwiss® are protected using the iOS Data Protection feature with the NS File Protection Complete class 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 SafeSwiss® does not perform background processing of push messages.
Note that the private key for the SafeSwiss® ID is stored separately in the iOS Keychain.

Android

SafeSwiss® 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 SQL Cipher 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.

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
Only the SafeSwiss® app can access this keychain entry.
While the entire iOS keychain is included in backups via iTunes or iCloud, When Unlocked This Device Only attribute causes the keychain entry of SafeSwiss® 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:

https://www.content.shi.com/SHIcom/ContentAttachmentImages/SharedResources/PDFs/Apple/apple-050521-AAW-Platform-Security-020221-ff.pdf

When the SafeSwiss® app is deleted and reinstalled, the keychain entry persists and the SafeSwiss® 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).

ADDRESS BOOK SYNCHRONIZATION

SafeSwiss® optionally lets the user discover other SafeSwiss® 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 SafeSwiss® IDs that have been linked with an email address and/or phone number. Any matches are returned by the server as a tuple (SafeSwiss® 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 SafeSwiss® 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 SafeSwiss® 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.

SafeSwiss® User Delete Account

SafeSwiss® users may delete their ID at any time within the application Settings > My Profile > Delete Account. 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.

CONCLUSION

1. **How private are my SafeSwiss messages?**
   All Safeswiss messages are transferred securely as well as locally protected with AES256 Elliptic Curve Cryptography (ECC) based algorithms. They can only be read by you and the recipients you sent the messages to and no one else.

2. **Does SafeSwiss offer full anonymity?**
   Each SafeSwiss user defines their own SafeSwiss ID and password. This is the only mandatory information required. A phone number or email address is **not** required to use SafeSwiss. This unique feature allows you to use SafeSwiss completely anonymously - unlike many other messaging apps.

3. **What makes SafeSwiss so secure?**
   SafeSwiss uses state-of-the-art asymmetric cryptography based on Elliptic Curve Cryptography (ECC) in general and on the 'Box' model of the NaCl Networking and Cryptography Library; to protect (encrypt and authenticate) messages between sender and receiver, as well as the communication between the app and the servers. SafeSwiss encryption code is open to independent audits.

   There are two layers of encryption: end-to-end layer between the conversation participants, and an additional layer to protect against eavesdropping of the connection between the app and the servers. The latter is necessary to ensure that an adversary who captures network packets (e.g., on a public wireless network) cannot even learn who is logging in and who they are sending a message to.

4. **What strength of encryption does Safeswiss use?**
   The asymmetric keys used in SafeSwiss 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.

5. **Can SafeSwiss or any authority read my messages?**
   That is absolutely not possible. SafeSwiss servers merely facilitate a secure exchange between sender and recipient. At no time is unencrypted message content stored on any SafeSwiss server.

6. **What user information is contained on SafeSwiss servers?**
   SafeSwiss servers are keeping registered SafeSwiss IDs as the only mandatory information. Any further data such as a user's profile pictures, email address or phone number is optional and only used for synchronization purposes. However, during the synchronization process all provided email addresses and phone numbers from your address book are anonymized (hashed) before they reach SafeSwiss servers. Once the comparison is finished, all data is immediately deleted from the server.

7. **Are SafeSwiss messages encrypted when they are stored on my device?**
   Yes, SafeSwiss includes its own, specific encryption based on XSalsa20 stream cipher as well as Poly1305 MAC authentication to protect stored messages, media, and your private key. The key used for this encryption is generated randomly the first time you start SafeSwiss. Please note: The Safeswiss application protection PIN which can be enabled independently is simply a UI Biometric lock and does not cause any additional encryption.

8. **What separates SafeSwiss from most other messengers with encryption?**
   Many providers of secure messengers claim that their product encrypts the messages that users exchange. However, most of the server operators can still read the content of the messages due to the following reasons:

   • Transport encryption only: usually only the connection between the mobile device and the server is encrypted, e.g., using SSL/TLS. While this means that messages cannot be intercepted while in transit over the network (a common problem in public Wi-Fi hotspots), they are in an unencrypted format once they reach the server.

   • End-to-end encryption without key verification by user: in this case, the provider claims that they utilize end-

to-end encryption, but due to missing user interface functions, the user has no way to verify that another contact's public key really matches with the private key that is only known to that contact. Therefore, it is relatively easy for an operator to perform a MITM (man in the middle) attack by manipulating the automatic key exchange without being detected. Subsequently, they can decrypt and even forge exchanged messages.

SafeSwiss uses state of art end-to-end encryption technology and enables users to verify the public keys of their conversation partners.

9. **What happens if for example authorities/law enforcement were to subpoena SafeSwiss servers?**
Safeswiss (or anyone) don't have the secret keys of our users (the secret key never leaves the device), there is no possibility to disclose any user related information. Safeswiss servers do need to know who is sending a message to whom, so that they can route it to the correct recipient, but they do not log this information, and cannot decrypt the message's content. Any conversation, video message, text, or phone teleconference is encrypted from your device to the other party's. Safeswiss or anyone has absolutely no access to it, so we can't disclose what we don't have access to.