GlobaLeaks Encryption Protocol

WHITEPAPER DRAFT NOTES

This white paper provides a technical explanation of GlobaLeaks's data encryption system.

Keys, Algorithms, and Data workflow 2

Users' Keys 3

Whistleblowers' Keys 4

Submissions' Keys 5

Encryption of questionnaires' answers 6

Encryption of comments 7

Encryption of messages 8

Encryptions of file uploads 9

Encryptions of files' metadata 10

Decryption of keys and and data 11

Key recovery 12

Keys, Algorithms, and Data workflow

Users keys are curve25519 keys;

Users keys are generated serverside at first user login.

Private users keys, were stored on the filesystem, are encrypted using symmetric encryption.

The symmetric key used for encrypting users keys is derived from the user passwordusing the KDF function Argon2ID. *
* The parameters for Argon2 used for KDF are stronger than the one used for authentication of related user which the hash is stored.
Default Argon2 parameters are currently defined to require 128MB per Login and 1 second of computation; This configuration is studied for mid sized servers and may be raised for more powerful servers (e.g. 1GB of ram per login).

Symmetric encryption is performed using XSalsa+Poly1305. *
* The implementation adopted is the SecretBox of pynacl (python binding to libsodium)

Asymmetric encryption is performed using curve2519+XSalsa+Poly1305 *
* The implementation adopted is the SealedBox of pynacl (python binding to libsodium).

Users' Keys

On first login a ECC Curve25519 key should be generated by the platform and saved on the User model by using:

- User.crypto_prv_key
- User.crypto_pub_key

From the ED25519 used for signing it is possible to derive a curve25519 key to be used for encryption.

Encryption KDF parameters should be by construction be always >= Authentication hashing

The key is generated:

- At first user login
- At password reset

Any time the user changes password, the system should use the user provided old_password to decrypt User.crypto_prv_key and the new user provided password to encrypt User.crypto_prv_key it again

Encryption should be based on XSalsa+Poly1305 using the key derived with scrypt from the user password.

Implementation API (change by decryption + encryption)

priv_key = symmetric_decrypt(encrypted_priv_key, passphrase)
encrypted_priv_key = symmetric_encrypt(encrypted_priv_key, passphrase)

Whistleblowers’ Keys

Key type: : ECC Curve25519

On submission a personal Whistleblower ECC Curve25519 key should be generated; The key should be saved inside WhistleblowerTip using:

- WhistleblowerTip.wb_prv_key
- WhistleblowerTip.tip_wb_pub_key

The private key should be stored encrypted with the system generate receipt following the same exact schema as 2.

Submissions' Keys

Key type: 128B: Xsalsa20-Poly1305

On submission, a per Submission 128B key should be generated and stored in:

- WhistleblowerTip.enc_key: encrypted with the WhistleblowerTip.wb_pub_key
- enkey in each ReceiverTip.enc_key: encrypted with the User.crypto_pub_key

Implementation API:

key = generate_symmetric_key()
the key should be encrypted/decrypted using symmetric_encrypt/decrypt as in 1

The key should be encrypted using users curve25519 key for each kind of users:
Recipients: the key is stored in the User table

- Whistleblower: the key is stored in the WhistleblowerTip table

Encryption of questionnaires' answers

On submission the submission should be encrypted and stored in:

- InternalTip.content: encrypted with the InternalTip.tip_pub_key

When encryption happens InternalTip.encrypted is set to True; this guarantee transparently migration for existing nodes where InternalTip.encrypted wil be false and the system will know that Submission answers will be directly accessible as well as (Comments, Messages, Files as described in next steps.)

Implementation API:

```
encrypted_data = symmetric_encrypt(data, key)
data = symmetric_decrypt(encrypted_data, key)
```

Encryption of comments

A new comment should be encrypted and stored in:

- Comment.content encrypted with the InternalTip.tip_pub_key

Implementation API:

Same as 5

Encryption of messages

A new comment should be encrypted and stored in:

- Message.content encrypted with the User.crypto_pub_key

Implementation API:

Same as 5

Encryptions of file uploads

A new file will be encrypted by using the tip_key stored in ReceiverTips and WhistleblowerTips.

As the software needs to handle large files bigger than the amount of RAM available in server and encryption without buffering is required but libsodium does not implement it natively, for our implementation of encryption in streaming the construction used is the one of dchest implemented in tweetnacl-js and MiniLock.

Encryptions of files’ metadata

Files’s metadata is encrypted by the system following the same construction of questionnaires’s answers.

Decryption of keys and and data

Decryption of each user (recipient, whistleblower) key is performed as the user login.
Any serialization of a Tip, Comment or Message should use the user private key to open the content specific key (per tips and comments) or the object directly (per messages).

Key recovery

The system implement a key recovery system by means of a recovery key.

Upon generation of a user key, the private key is symmetrically encrypted with a recovery key randomly generated.

For usability reasons., this recovery key is as well kept asymmetrically encrypted with the public key of the user for making it possible for a logged user, in possession of its password/key, to retrieve the recovery key from the system.