GlobaLeaks Browser PGP Encryption

(Protocol Specification 8.7.2016)

Table of Contents

GlobaLeaks Project by
Hermes Center for Transparency and Digital Human Rights
http://logioshermes.org - https://globaleaks.org

Goal

With GlobaLeaks now in widespread use among many successful initiatives, the Hermes Center intends to make it easier for people to stay safe while using the platform.

One of the largest barriers to using the platform safely is the use of GPG for the encryption and decryption of files submitted by whistleblowers to a server running GlobaLeaks. Most organizations that use the software opt out of PGP encryption due to the added complexity it places on the journalists handling encrypted material.

As a result the Hermes Center has produced this specification with two related goals in mind. Improving the system's usability for the average user and improving the security of the system for these users.

Introduction

The changes to GlobaLeaks described here move data encryption of off the server and into the user's browser. This stops sensitive data from reaching the server in plain text and it moves key generation and decryption from a program like gpg into the recipient's browser.

By using a JavaScript implementation of OpenPGP, messages and files can be sent between a whistleblower and their intended recipients hassle free.[1]

This specification describes the cryptographic components released in GlobaLeaks-2.99.0-unstable. This work and the accompanying implementation is a core component of an Open Technology Fund grant for 2015 and 2016.

Related Work

The motivating example for this work is Hushmail's end-to-end encryption scheme.[2] The requirements are clearly different, but the principles remain the same.

Unlike Hushmail's design for an cryptographic system in a web browser, this implementation does not store any state in a user's browser. While significant trust is placed on the server to provide the proper keys and messages for users, the client does not preserve state across sessions. This choice was made to preserve a whistleblower's ability to plausibly deny their use of the system.

[1] See OpenPGPjs

[2] Hushmail Whitepaper circa 2001

Document Structure

This first section describes at a high level the requirements, the functional specification of the new protocol and the changes to the threat model as a result.

The second part describes in detail the new external libraries and the chosen parameters used in the implementation of the protocol.

This structure lets us consider the design of the system and the chosen parameters independently. Tweaking parameters and tightening the design will be done after a public review is conducted.

Requirements

1. Recipients, administrators, and users in general must only remember a username and password.
2. A whistleblower must only keep secret a short alphanumeric string.
    a. A whistleblower should not have to save any files on their machine
    b. The server should never learn the whistleblower's secret.
    c. Using the client should leave as few forensic traces as possible on the whistleblower's computer.
3. A user should only need a web-browser to use the tool.
    a. GlobaLeaks should leave nothing in local storage or on the device.
    b. GlobaLeaks should be responsive and functional even on memory and CPU constrained devices.
4. Existing GlobaLeaks platforms should be able to upgrade to the new system.
    a. The upgrade must not erase existing data.
    b. The upgrade should not require a special update process for the users.

Functional Specification

Conceptually, the protocol is simple. A GPG key pair is generated for every kind of user – whistleblowers, recipients, administrators. Every message exchanged between users is encrypted with the corresponding public key of the intended recipients. There is no authentication of encrypted messages, and the public keys supplied by the server are trusted as the keys of the intended recipients for those messages.

Specifically a client will now:

1. Generate a key pair.
2. Encrypt every message sent between a whistleblower and a recipient with both public keys.

3. Encrypt every comment and the whistleblower's response to the initial questionnaire with the whistleblower's and the recipients' public keys.
4. Encrypt every file with the public keys of the intended recipients.
5. Decrypt in a similar fashion.

New Threats

For an attacker that wants to discover the identity of receivers or execute code on a receiver's machine these changes introduce new vectors. Since the browser is now performing many different cryptographic operations in Javascript the risk of leaking essential information about a whistleblower or a receiver is increased.

A defect in OpenPGPjs, the WebCrypto functions implemented by the browser, or memory isolation of the browser or operating system could lead to an attacker recovering the user's private key material. With the private key an attacker could read all of the messages and files sent to a user. Thus the browsers supported by the system must be robust.

An attacker that physically seizes the GlobaLeaks server will find the relevant content of the node encrypted. The attacker will also find the passphrase protected keys used to encrypt that content. Thus the passphrase derivation used to encrypt those keys must be robust enough to stand up to offline attacks.

Protocol

Terms

A user's key_passphrase is defined by:

```
key_passphrase = scrypt(password, user_salt)
```

An auth_token_hash for users is defined by:

```
auth_token_hash = sha512(key_passphrase)
```

A whistleblower's receipt is a 16 digit number created by a client for the whistleblower at the time of submission. The value receipt is shown to the whistleblower after they have successfully created a submission.

A whistleblower's key_passphrase is defined by the following:

```
key_passphrase = scrypt(receipt, user_salt)
```

A reciept_hash for Whistleblower's is defined by:

```
reciept_hash = sha512(key_passphrase)
```

User Creation

Every new account created on the platform is initialized with the configurable default_password and user_salt chosen by the server.

Upon the first successful authentication an account-holder is required to select a new password and generate their key pair. This operation is identical to the password change operation with the addition of the user's public key.

Authentication

The protocol for authentication works in four steps:

1. Client submits username
2. Server responds with user_salt
3. Client computes auth_token_hash and sends it to the server
4. Server checks auth_token_hash against the user's stored value and sends a session token to the client

Whistleblower Authentication

A whistleblower authenticates in three steps

1. The server sends the client node_salt
2. The client computes recipeit_hash and sends it to the server
3. Server checks receipt_hash against all of the stored receipt hashes and returns the whistleblower's wb_private_key.

Authentication Token and Key Update

The protocol for changing a user's password requires that both the auth_token_hash and the enc_private_key are stored on the server.

To change a user's password the client must compute:

```
old_auth_token_hash = sha512(scrypt(old_password, user_salt))
key_passphrase      = scrypt(new_password, user_salt)
new_auth_token_hash = sha512(key_passphrase)
enc_private_key     = encrypt(private_key, key_passphrase)
```

The client then submits the following to the server, where public_key is set only when a user has never submitted a key to the server before:

```
{
  old_auth_token_hash,
  new_auth_token_hash,
  enc_private_key,
  [public_key]
}
```

If the input passes the server's normal validation checks, the request has a valid session token, and old_auth_token_hash equals the server's copy of auth_token_hash, the server will use new_auth_token_hash to authenticate the user and serve enc_private_key to the user's client from now on.

Submissions Creation and Encryption

When the client of a whistleblower begins a submission it generates a key pair and then computes key_passphrase and receipt_hash.

Upon submission, the client encrypts the answers to the questionnaire the whistleblower filled out and the files the Whistleblower attached using the public keys of the intended recipients.

The format of the message sent to the server is as follows:

```
{
  receipt_hash,
  wb_enc_private_key,
  wb_public_key,
  enc_answers
}
```

When the server receives the submission, metadata like the time of submission, intended recipients, and the transport method used are stored in cleartext.

Whistleblower Submission Access

A Whistleblower can access a submission they have created by providing their receipt. The client then authenticates with receipt_hash.

By deriving key_passphrase the client is able to decrypt and use wb_private_key to decrypt any messages or comments a recipient has sent to them.

By default the Whistleblower's access to their submission is temporary. The server will prevent their access to the submission after 15 days and delete wb_private_key and receipt_hash but will continue to store the whistleblower's responses and messages.

Message and Comment Encryption

Each comment made on the submission is encrypted with the keys of all the recipients and the whistleblower. Each message that the whistleblower sends to a recipient is encrypted with the recipient's key and the whistleblowers or vice-versa .

Password Reset

To handle the case when a user loses their password an admin can reset the user's credentials. In this scenario, all of the user's encrypted material is considered lost, so all of the encrypted material for that user is deleted, the user's key pair is deleted, and the user's auth_token_hash is reset to sha512(scrypt(default_password, user_salt))

Appendix

Whistleblower Receipt Format

The Whistleblower receipt is a 16 digit decimal number. This input space is too small to provide strong protection against an attacker that wants to compute every possible key_passphrase used by the Whistleblowers.

For this reason further analysis is needed to keep the size of the input space small, yet still provide reasonable brute force protection.

New Client Dependencies

- Version 2.3.2 of OpenPGP.js - used for all operations related to OpenPGP in the client.
- Version 1.2.0 of async-scrypt.js - used for scrypt
- The Candidate Recommendation of the Web Crypto API - used for the getRandomValues function.

Key Pair Structure

Each PGP key contains a 2048 bit RSA Sign-Only key and a 2048 bit RSA Encrypt-Only key. Before the secret key is stored on the backend the key_passphrase is used as input to encrypt the secret key bits using OpenPGP's iterated and salted String 2 Key algorithm[3].

The secret key packets have the following format:

[3] See section 3.7.1.3 of RFC 4880

```
:secret key packet:
     version 4, algo 1, created 1467897201, expires 0
     skey[0]: [2048 bits]
     skey[1]: [17 bits]
     iter+salt S2K, algo: 9, SHA1 protection, hash: 8, salt:
ce79a59ce5af7789
     protect count: 65536 (96)
     protect IV:  92 45 99 d0 4a 6e 1a 68 ca 1c 75 68 cf 20 96 7b
     encrypted stuff follows
     keyid: 17619A92F2FF6B38
```

Selected Algorithms

- RSA Encrypt-Only is the only algorithm used to derive the shared key.
- AES-256 with OpenPGP's CFB is the preferred symmetric encryption algorithm in the system

Scrypt Parameters

As a requirement for maintaining compatibility with current deployments of GlobaLeaks, the following parameters of scrypt are used.

```
N = 14, P = 8, dkLen=64
```

The choice of these parameters represents a desire to maintain backwards compatibility with existing deployments of GlobaLeaks. They are subject to change.

Migration Handling

Given the scrypt parameters chosen above it is possible to migrate an existing GlobaLeaks server to the new scheme in one migration. After this operation all existing submissions on the server will be marked as old and subject to expiry. These existing submissions will not be encrypted and only support a simple comment to inform the whistleblower of closure of a submission.

The old variable password is stored as scrypt(password, user_salt) in the old database. This value is identical in structure to key_passphrase and thus can be used to compute auth_token_hash.

When users authenticate to the upgraded system for the first time they will perform the password change operation described above in Password Reset.