Stellarport stellar-keystore Audit Report

Dmitry Chestnykh

February 26, 2018

Introduction

This document describes the results of the audit of code located at

`https://github.com/stellarport/stellar-keystore/blob/91c48341df65c1dc437e2decc61337fb7af2b8bf/src/index.js`

Description

The goal of *stellar-keystore* is to provide a user with a downloadable file containing a private key generated by Stellar SDK [1] encrypted with a user-supplied password. The file can then be supplied to the program along with the password to restore the private key.

A password supplied by the user is stretched with *scrypt* sequential memory-hard key derivation function with a randomly generated 32-byte salt, resulting in a 32-byte secret key, which is then used along with a randomly generated 24-byte nonce to encrypt and authenticate the private key using *XSalsa20-Poly1305* (NaCl *secretbox*). The salt, the nonce, *scrypt* parameters and the resulting ciphertext, along with metadata, are then provided to the user in a JSON-encoded file.

To restore the private key, the salt and *scrypt* parameters from the file are used along with the user-supplied password to derive the secretbox key, which is then used along with the nonce from the file to authenticate and decrypt the ciphertext.

The default *scrypt* parameters are hard-coded to N=16384, r=8, p=1, which add approximately 19 to 25 bits of security to passwords [4]. Considering the pure JavaScript implementation, such parameters allow using the library on a wide range of devices, including low-performance and low-memory smartphones. Since *stellar-keystore* stores the parameters in the file, it allows increasing the default ones in the future in a backward compatible way.

Implementation uses *scrypt-async-js* [2] and *tweetnacl-js* [3] JavaScript packages ported and/or implemented by the author of this audit report.

Results

The audit showed that the algorithm is implemented correctly and securely.

Observation

When loading *scrypt* parameters from untrusted sources it is useful to limit them to reasonable values to avoid denial-of-service attack on the client (for example, by setting the unreasonably high value of N) by refusing to derive keys if they are above the limit. However, for this library such attack is meaningless, since 1) the user controls the file, 2) the JavaScript runtime limits the amount of memory it can allocate, throwing an exception if the limit is exceeded. If in the future this library will be used for storing keys on third-party remote location that is not under the user's or Stellarport's control, it is recommended to implement such limits.

References

1. js-stellar-sdk: https://stellar.github.io/js-stellar-sdk/
2. scrypt-async-js: https://github.com/dchest/scrypt-async-js
3. TweetNacl.js: https://github.com/dchest/tweetnacl-js
4. Mailing list exchange: http://mail.tarsnap.com/scrypt/msg00247.html