# Standard Notes

Standard File Cryptography Design Review

May 22, 2017 *Updated August 5, 2017*

## Table of Contents

## Project Details

In May of 2017, Standard Notes (Mo Bitar) requested that Shackle Labs conduct a design review of the use of cryptography in the Standard File specification, this report represents the results of this effort. Shackle Labs has taken all reasonable efforts to complete this cryptography design review using industry standard techniques and technologies and staff with expertise in the field, to ensure the quality and accuracy of this report.

### Project Overview & Objectives

At the request of Standard Notes (Mo Bitar), Shackle Labs has completed a thorough design review of the Standard File specification's use of cryptography.

The goal of this project was to determine the if flaws exist in the use of cryptography that would lead to a loss of confidentiality or integrity. Such flaws represent a significant risk to the users of the Standard File specification, including Standard Notes.

The specification version reviewed was v0.0.2, retrieved from https://standardfile.org/ on May 17, 2017.

This effort began on May 17, 2017 and was completed on May 22, 2017.

On June 29, 2017, at the direction of Mo Bitar, an updated version of the Standard File specification was reviewed (v0.0.2, retrieved June 29, 2017) to determine if the issues identified in the initial audit were remediated. All findings were reviewed (recommendations were not reviewed), findings 1, 3, and 5 were found to be remediated, finding 2 was closed after discussions with the client and a change in documented purpose, finding 4 remains open.

### Project Team

The following representatives of Standard Notes (Mo Bitar) took part in this effort:

- Mo Bitar (Primary Contact)

The following members of the Shackle Labs team took part in this effort:

- Adam Caudill, Chief Research Officer
- Taylor Hornby, Consultant

### Scope & Limitations

The scope of this project is defined as only the portions of the Standard File specification related to cryptography, no API interface or other portion of the system was examined.

The following limitation should be noted, as they were out of scope or due to technical, legal, or other reasons had to be excluded:

- No implementation of Standard File, including Standard Notes was examined, this design review was limited to the specification only. Additional issues are likely to exist if an implementation does not adhere to the specification.
- The specification includes the use of JSON Web Tokens, the use of these tokens was not included in this review.

## Executive Summary

At the request of Standard Notes (Mo Bitar), Shackle Labs has completed a design review on the use of cryptography in the Standard File specification. For Standard Notes (Mo Bitar), this effort was led by Mo Bitar, and for Shackle Labs, Adam Caudill. At the direction of Mo Bitar, the scope of this effort was defined as only the portions of the Standard File specification that relate to cryptography; no implementations were reviewed. Based on a thorough assessment of the Standard File specification, Shackle Labs has found the following.

Five exploitable issues were found, with one being rated as High severity, in addition, there were eight informational findings.

The most severe issue found is that, per the specification, the client will blindly trust the Key Derivation Function (KDF) parameters supplied by the server. This allows a malicious or compromised server to weaken the parameters to the weakest that the client supports, making it far simpler for the server to recover the user's password. Given the specifications stated goal of minimal trust in the server, this violates that goal and introduces a substantial risk of attack.

It was found that there is no cross-application isolation, which can lead to a malicious or compromised application to steal all data belonging to a user, including data stored by a different application. It was also found that there is no cryptographic binding between the name and content of a file, allowing a compromised or malicious server to swap the content of a file with that of a different file, and the client would be unable to determine that this switch had occurred.

Two Low severity issues were also identified, relating to the server's ability to transparently delete user data, and an error in the specification that could lead to erroneous implementations. Eight informational findings were identified, some of which would improve the security of the specification, some of which address likely vulnerabilities in implementations of the specification (which were not tested in this review).

Shackle Labs strongly recommends that a new version of the Standard File specification be created that addresses these issues as soon as possible. While overall, it was found that the Standard File specification was well defined and shows that significant effort was placed into meeting industry standard best practices, significant issues were found that should be addressed.

## Findings

During the course of this review, the following was found – these issues are rated based on two criteria:

- Severity: The impact of the issue, if exploited.
- Exploitability: The ease with which the issue can be exploited.

In addition, informational findings are also included, that would increase the security of the design or quality of the specification.

## #1 – Client Blindly Trusts KDF Parameters Sent from Server

Severity: High
Exploitability: High
Status: Remediated

Updated July 2, 2017: This issue was fixed as recommended by fixing some of the PBKDF2 parameters (hash algorithm, output length) and authenticating the others (salt, iteration count).

Updated August 5, 2017: A security flaw in our original recommendation to authenticate the pw_cost and pw_salt parameters was reported by Dmitry Chestnykh. The section below has been updated to remove the vulnerability. A discussion of the error can be found in the Errata section.

Quoting the project's design document...

> *Standard File attempts to make no final judgement on the best key derivation function to use, and instead defers to clients to make this decision. This allows for a future-proof implementation that allows clients to adjust based on present-day security needs.*

...explains why the parameters `pw_func`, `pw_alg`, `pw_cost`, `pw_key_size`, and `pw_nonce` are stored on the server. These parameters are not authenticated, so the server is free to weaken them in an attempt to break the user's security.

- `pw_func`, `pw_alg`: If the client supports multiple KDFs or hash algorithms, the server can force the client to use the weakest ones. This is a "security usability" problem: the more options there are and the more untrusted input the client is receiving from the server, the harder it is to implement the client securely. It would be better to enforce a specific algorithm choice in the spec that can only change in later versions of the entire protocol (e.g. PBKDF2-SHA256 for now).
- `pw_cost`: The server can set this to a low value to make it easier to brute force the user's original password from the value that gets sent in the POST `auth/sign_in` request. This parameter should be compared against a minimum value.
- `pw_key_size`: The client does not authenticate this value, and so the server can exploit it in either of two ways. Firstly, if the server sends a value of 0, then the client will use zero-length keys to encrypt all future files and modifications to files. Secondly, if the server sends a value which is the double of the correct value, then the `pw` value sent to the server will contain the master key. This happens because of a property of PBKDF2: if you ask for a longer output, the leading part does not change, so if say the original `pw_key_size` is 32 and the server sends 64, then the computed `pw` will be the same as the `pw||mk` that's computed in the non-attack case. To fix this, all key lengths should be fixed for the version of the protocol and only changed in later versions of the entire protocol.

- `pw_salt`: The server can set this to an all-zero (or constant) value, making it easier to brute force the user's original password from the value that gets sent in the `POST auth/sign_in` request.

The `pw_func`, `pw_alg`, and `pw_key_size` parameters should be removed and made part of the protocol specification. The pw_cost parameter should be checked to be no less than a minimum value, say 100,000 iterations. The pw_salt should be derived from a random pw_nonce value stored on the server combined with the user's account identifier, for example,

pw_salt = SHA256(email + "#" + pw_nonce).

This should be computed on the client to prevent the server from obtaining hashes of different users' passwords under the same salt. The client should check that the length of the pw_nonce they receive from the server is some standard value defined in the protocol, say 256 bits.

## #2 – No Cross-Application Isolation

Severity: Medium
Exploitability: High
Status: Closed

Updated July 2, 2017: As far as we can tell, this issue has not been fixed. Enforcing boundaries between clients will require either (A) the user giving different strong passwords to each client or (B) implementing something OAuth-like and trusting that the server will enforce access controls.

Per discussions with the client, Mo Bitar, Standard File is no longer intended to be used by any application other than Standard Notes, and as such, this issue is deemed to be closed, but not remediated.

All applications the user logs in to derive the same master key, and have access to all of the same data.

This means that if the user accidentally logs in to an application that is untrustworthy (or simply less trustworthy than any of the other applications the user is using), that application can steal their master key and all of their data.

To fix this, applications need to be given different keys. Fixing this requires re-architecting the protocol, since as it stands applications are provided with the email and password which is enough information to impersonate the user. If this is not fixed somehow, then both application developers and the user should be aware that all applications are able to access to **all** of their data (even other applications' data).

## #3 – No Cryptographic Binding Between Name and Content

Severity: Medium
Exploitability: High
Status: Remediated

Updated July 2, 2017: This issue was fixed by including the UUID in the HMAC calculation, and checking the UUID upon decryption. This will prevent the server from swapping the contents without also swapping the UUID.

The sentence...

> "Given a string_to_encrypt, an encryption_key, and an auth_key:"

...should be changed to...

> "Given a string_to_encrypt, the item's UUID, an encryption_key, and an auth_key:"

...to make it easier to understand where the UUID in that section of the document is coming from.

There is nothing cryptographically binding a (`content`, `enc_item_key`) pair to the name of the content (the UUID). Without this, a server can freely swap the contents of files without being detectable by the user.

For example, if the user has a document they are about to share with the public and a document they wish to keep secret, the server can swap their contents at the last minute to have the user publish the secret document by mistake. Or, if the user is a malware researcher storing malware samples in Standard File as well as backing up their system to Standard File, the server could replace some executables in their system backup with the malware.

To fix this, the ciphertext needs to be cryptographically tied to the UUID, e.g. by including the UUID in the HMAC calculation of `auth_hash`.

## #4 – Deleted Bit is Unauthenticated

Severity: Low
Exploitability: High
Status: Open

Updated July 2, 2017: As far as we can tell, this issue hasn't been fixed. It's a minor issue so it seems like the most appropriate thing to do would be to add a parenthetical remark "(not cryptographically authenticated)" to the field's description in the table listing the item model properties.

The deleted bit is unauthenticated, so the server may delete files without detection by the client. Depending on what kind of application is running on top of Standard File, this might have bad security consequences. For example, if the application makes meaningful use of the "deleted" or "not deleted" status of a file beyond just deleting things to free up storage space.

This might not need to be fixed, but in any case, the problem should be communicated clearly to application developers.

Write a threat model document describing precisely what security guarantees Standard File is intended to provide. For example, mention that the server can delete files, that the server can restore files to earlier versions, that the server can change the `content_type`, and so on.

These issues must be communicated clearly to application developers otherwise they might rely on security features they assume must exist but actually don't.

## #5 – Inconsistent HMAC Parameter Ordering

Severity: Low
Exploitability: Low
Status: Remediated

Updated July 2, 2017: This issue has been fixed. All mentions of "HMAC" in the document now use the (data, key) parameter order. This is consistent with the order in RFC 2104.

At one point in the design document, the key is passed as the first parameter to HMAC,

```
encryptionKey = HMAC-SHA256(mk, "e")
...
authKey = HMAC-SHA256(mk, "a").
```

However, at a later point, the opposite parameter ordering is used,

```
local_auth_hash = HMAC-SHA256(string_to_auth, auth_key).
```

Using the wrong parameter order could lead to security problems. Fix this by using a consistent parameter ordering everywhere in the document. The HMAC RFC (RFC 2104) uses the (data, key) order, so that order is probably best.

## Recommendations

The following are informational recommendations to improve the quality and security of the specification.

### Support Two-Factor Authentication from The Start

If an adversary ever learns the user's password, they can steal all of the user's data as fast as they can download it from the server. This is especially important for Standard File, since the user will be accustomed to giving the same username and password to many applications, and so will be more susceptible to phishing attacks. It would be beneficial to build in support for time-based one-time-passwords to mitigate this risk.

### Rename "password" to Something More Appropriate

The parameter sent from the user to the server during authentication is called "password" even though it is not a password. Even though there are plenty of warnings, the application developer might get confused and think they are supposed to send the user's actual password. The parameter should be renamed to something else, and the server should perform a strict length check on the value to make sure any application that is making this mistake by accident won't work.

### Prevent Future Version Rollback Attacks

Clients are told to decrypt ciphertexts in the old (`001`) format using the old protocol automatically. Since older versions might contain vulnerabilities that newer versions don't have, version rollback attacks are a common problem. New clients that only create and use files in the newest version should not fall back to decrypting files in older versions. Clients supporting newer versions should also re-encrypt files with the newest version when they next modify their contents (the server might even be able to enforce this).

### Standardize on One Hash Function

The protocol makes use of SHA1, SHA256, and SHA512. There is no good reason to require the client to have an implementation of all three of these hash functions, just use SHA512 everywhere.

### Suggest Comparing MACs in Constant Time

The comparison between the HMACs `local_auth_hash` and `auth_hash` must be done in constant time. Most developers aren't aware of this subtlety and will probably just use their language's built-in "==" string comparison.

It might be worth mentioning that the comparison should be done in constant time.

## Write a Threat Model

As suggested in Issue #4, a threat model should be written so that applications using Standard File know which security properties are provided and which aren't. Otherwise they might think Standard File provides some security guarantee when it actually doesn't.

## Errata

An earlier version of this report recommended an erroneous fix to the unauthenticated pw_cost and pw_salt parameters in Finding #1. The original recommendation was as follows:

> To authenticate `pw_cost` and `pw_nonce`, change the protocol so that the authentication steps look something like this:
>
> ```
> key = pw_function(uip, pw_salt, PBKDF2-SHA256, 768, pw_cost)
> ak = key.substring(64, 32)
> local_mac = HMAC-SHA256([pw_salt, pw_cost].join(":"), ak)
> if timeSafeCompare(local_mac, pw_auth):
>     pw = key.substring(0, 32)
>     mk = key.substring(32, 32)
> else:
>     raise an error
> ```
>
> Where `pw_auth` is a new parameter stored by the server and `timeSafeCompare` is a timing safe string comparison.

We are grateful to Dmitry Chestnykh for noticing that the pw_auth parameter is effectively a password verifier – it can be used to check guesses of the user's password in an offline attack – and that the server will provide it to anyone who knows the account's email address. This puts users with weak passwords at a greater risk than if the server had limited control over the salt, so we have updated our recommendation.