Susi Software Specifications

Sean Francis N. Ballais <sean@seanballais.com>

March 4, 2024 11:56 PM

Table of Contents

Disclaimer

This specification document is a living document. It is continuously being updated to reflect the latest developments of Susi. This document may already be outdated. Please contact Sean Ballais <sean@seanballais.com> to request the latest version of this document.

1. Overview

Each one of us has files that we see as critical and sensitive. These files can range from financial documents to backup codes. However, keeping these files in raw, unencrypted form puts their data at risk of being stolen or viewed easily. Cloud storage services increase these risks as well. These services increase the number of ways our data can be obtained without our permission. While these services have high-security standards, files stored in such services are not always guaranteed to be protected from theft as well. Most services store files in unencrypted form (and only during transit), making them easily readable by third parties.

Susi is built to provide *one* answer to this problem by locking/encrypting files and only opening them for a limited amount of time or until they are no longer being used. The initial version of this project can be completed in less than a month.

1.1. Goals

Our goal for Susi is to protect sensitive files and prevent unauthorized access to them with as few obstacles as possible to the user.

1.2. Product Requirements

- Target files are encrypted securely, and can only be decrypted by the password set by the user upon encryption.
- Encrypted files must carry the hash of the original file to help with ensuring that the decrypted file is the original file.
- The encrypted file must not be present when the decryption is successful. Consequently, after encryption, the original/decrypted file must no longer be present.
- If encryption fails, the original file must remain. Naturally, if decryption fails or the hash of the original file is different from the decrypted file, the encrypted file must remain.
- Decrypted files must remain decrypted while they're in use. However, once they're no longer in use for five minutes, they must be encrypted again.
- Allow files to be encrypted and decrypted from the file manager.
- Susi will open on startup.
- Logs are written by the app to help with debugging and behaviour tracking.
- Encrypted files have a file extension of `.ssef`.

1.3. Assumptions

- We are only targeting Windows 10 and 11 for now. We will add support for macOS and Linux at a later time.
- Only files will be encrypted.

1.4. Out of Scope

- Susi does not aim to prevent the copying of protected files. What it does is simply prevent access to the data in these protected files.
- Decrypted files will also be vulnerable to copying currently.
- Folders will not be encrypted.

1.5. Open Questions

There are some considerations that are not completely resolved yet, and would require some inquiries and even testing.

1. Should we also allow encrypting/locking folders?
2. Should we compress encrypted files?
3. Does byte alignment help improve performance?

2. Approach

2.1. Components

Major Components

There will be two major components for Susi: (a) Susi Core, and (b) Susi GUI.

Susi Core is the core application that has the following responsibilities and behaviours:

- Monitors protected files and encrypt them using a password when they are no longer in use after 5 minutes.
- It is responsible for decrypting files.
- Runs in the background.

On the other hand, Susi GUI is be assigned to handle user interactions. It has the following requirements:

- It will pop up during encryption and decryption to request for the password
- When a message needs to be displayed, such as when encryption fails, it will display a notification bubble to be shown to the user.
- Responsible for displaying the window that orients the user when running the app for the first time.
- Responsible for integrating with the OS's file manager.

Communications between Susi GUI and Susi Core is done via IPC.

Susi Core is written in Rust, while Susi GUI will is written in the "standard" language for writing GUI apps of the target operating system. In Windows, this is C#. In Linux, this is C or C++. For macOS, it is Swift.

File Format

The file format for an encrypted file (`.ssef`) consists of three parts: (a) the file identifier, (b) the metadata, and (c) the encrypted file data. The version of the file format here is designated to be v1. This file format is designed to be compatible with any version of Susi that ~~is loyal to~~ supports v1 of this format.

The file identifier helps us determine if the encrypted file is a valid `.ssef` file. It takes the first 4 bytes of an encrypted file, and has the following format:

```
[0x55 0x3F]
[(format version, 2 bytes)]
```

The first two bytes is the file identifier, and the next two bytes specify the format version. The

likelihood of this format reaching a version greater than 65,535 (2^16^ - 1) is low, as such, we can safely allocate two bytes for the format version.

The next part is the metadata. This contains any metadata we choose to add to the file that we can use during decryption. Most of the metadata is expected to be key-value pairings. This part has the following format:

```
[(length of metadata in bytes, 2 bytes)]
[(metadata ID, 2 bytes) (metadata value length (MIL), 2 bytes) (metadata content,
length specified by MVL)]
[more metadata items here...]
```

In the metadata section, the first two bytes specify the length of the section. This gives the section a maximum of 65,536 bytes, with 65,534 bytes of it being usable. The next bytes are the metadata key-value pairs, also referred to as metadata items. The first four bytes contain the key of the pair. The first two bytes of the key contains the ID. This ID specifies what type of metadata the item is holding. Currently, we support the following IDs:

Metadata ID	Description
`0x00 0x01`	The original **filename** of the encrypted file, including the file extension.
`0x8A 0x58`	The **hash** of the original, unencrypted file. Used to ensure that the decrypted file is the same as the original file. The ID is meant to be read as "hash", with 'H' being replaced with '8' since 'H' is unavailable in hexadecimal and there is no known one-symbol Jejemon or Leetspeak substitution for 'H' that uses a symbol that is valid in hexadecimal. So, we opted to use the ordinality of 'H' in the English alphabet, which is 8th, instead, since the number is a valid hexadecimal symbol.

The next two bytes in the key specify the length of the metadata value in bytes. For easier communication, this length will be referred to as the MVL. The next MVL bytes will contain the value of the metadata item.

Another metadata item may placed be after these bytes. We can also add padding bytes, denoted with `0x00`, to help with byte alignment. These paddings may help improve performance when decrypting files, but that remains to be tested.

If the specified length of the metadata section has already been reached, the encrypted data will be reached. This data is the last section of the file and will span until the end of the file. It will be the data we will be decrypting and saving into another unencrypted file.

2.2. Security and Privacy

Encrypting and decrypting files does present the risk of losing that file forever. One case is when the file cannot be decrypted by the app. Another would be when the app, due to a glitch, accidentally deletes the file during an encryption or decryption process. Additionally, the file is vulnerable while it is decrypted and can be copied. Files may also be stolen during transit or at rest. As such, these files can be decrypted offline. This case cannot be completely solved by Susi, which

can only protect the file through encryption. In the worst case, Susi may be used by malicious third parties for their own gain. Files can be encrypted by anyone and can be held as hostages. These risks must be clearly communicated to the user, particularly during installation and first run.

On the privacy side of things, there are no concerns. The app is not expected to cause any significant impact on the user privacy. The only time the app can risk the privacy of a user is when a third party successfully decrypts a user's protected file using the app, uncovering the secrets of a user deep within the file.

Susi must be written so that it does not result in the loss of valuable files. It must make sure that deletion of files will only occur when the encryption or decryption process has already succeeded.

2.3. Test Plan

For testing, we utilize automated testing (unit tests and integration tests) to help us ensure that the code performs as expected and reduce the risks of losing and leaking file data. We utilize GitHub Actions to automatically run these tests on every commit and pull request.

Manual testing is also utilized and select volunteers can use Susi in their own environments. Having volunteers helps us find bugs and issues that have slipped through our internal testing. Dogfooding is another method we utilize to help us find problems with the app.

3. Operations

3.1. Deployment and Rollout

Anyone may request access to the app by contacting Sean Ballais. A download link to the app will be provided. The download link may have a time limit, however. If Susi becomes feasible to be sold to third parties, it will be made available from the website of Sean Ballais. When we deploy for release, a GitHub Action is used to automatically build the project and generate the setup files. The distributable files will primarily be made available via Humble Bundle, with users purchasing copies through the Humble Widget.

3.2. Rollback Plan

If there are any issues with the currently released versions of the software, such as a non-working installer or critical bug that slipped through testing, the affected versions will be retracted. If the current version has a critical issue, we will revert back to the previously known working version of Susi. Users will be informed of the change via social media. We may also contact users via email if we have their email addresses.

3.3. Monitoring and Logging

Susi will log any important information, warnings, or errors. These logs may then be voluntary uploaded by the user for whatever purposes, such as when attempting to fix any issues the user has encountered.

If we use the Humble Widget, we will be granted access to real-time sales analytics provided by Humble Bundle.

3.4. Metrics

Susi is considered to be a success when it has accomplished all product requirements have been completely satisfied. A secondary success metric is knowing at least 5 people are using Susi regularly. Being able to sell the software to others will be considered to be bonus points and a completely optional metric to achieve.

3.5. Long-Term Support

Susi will continuously be maintained by Sean Ballais for the foreseeable future. It will be maintained on a free-time basis. If there are a sizable number of users that find the app useful and use regularly, especially when selling the app is found to be feasible, more resources will be allocated in improving the app.

4. Timeline of Development

1. March 5 - 10, 2024:
    a. Development of Susi Core
2. March 11 - 16, 2024:
    a. Development of Susi GUI
3. March 17, 2024:
    a. Distribution to friends for regular usage and testing.
        i. **Note**: Pray and hope that none of their machines explode in a fiery fashion.