Review of ScramFS secure file system

Vanessa Teague (`vjteague@unimelb.edu.au`)
School of Computing and Information Systems
The University of Melbourne
October 16, 2017

1 Introduction

This is a brief review of the ScramFS secure file system architecture. ScramFS is a secure cryptographic file system developed by Scram Software, who engaged me to review the security of its design from a cryptographic perspective. I have read the documents and accompanying security analysis with care. As far as I can tell, the system is cryptographically sound. The design and the documentation are both of a very high standard, with rigorous security definitions and detailed proofs of the security properties.

However, reviews like this should always be interpreted with caution, because even the most carefully-studied standards (such as TLS) have been shown to have subtle errors. This report should not be interpreted as any sort of guarantee that the system has exactly the security properties it claims to have, or that the aspects I reviewed have no errors, merely that a careful analysis over about 16 hours did not detect any errors.

As requested by Scram Software, my review focused on the cryptographic aspects of the file system and did not consider the other (non-cryptographic) parts of the design. Specifically, I looked at the high-level design and technical overview (Sections I and II), the low-level cryptographic primitives (Level 0) and the private functions (Level 1) comprising key derivation, encryption and authentication of file/directory paths, file headers, file footers and file content. I have not looked at the string and system functions (Level 0) or the high-level file system functions (Level 2), such as `open()`, `read()`, `write()`, `listdir()`, `makedir()`, nor the implementation.

If there were errors in the other levels, this would undermine the security of the system. One critically important assumption is the input of good quality randomness into the key generation process. This detail must be very carefully implemented—if the initial value in the key generation process has insufficient entropy then there is no security guarantee.

I encourage the authors to make this report, the security analysis, and the details of the system openly available so that any errors or weaknesses can be found and corrected. A bug bounty program (like Google's or Facebook's) could offer a small reward for responsible disclosures of errors.

This report summarises the system's security properties, with the aim of making clear what the assumptions and implications are.

2 Summary of the main security properties

ScramFS uses well-chosen cryptographic primitives—I did not find any errors in either their choice or use. AES is the chosen symmetric encryption scheme, in a variety of configurations that provide both encryption (secrecy) and authentication (defence against manipulation). Their security properties are based on published work and the probability of successful attack is carefully quantified.

AES-256 has been chosen, though 128-bit keys would be sufficient now, in order to defend against the possibility of a successful quantum computing attack.

The security model allows for a very powerful attacker who can control all of the storage as well as the inputs to the ScramFS client. This models an attacker who compromises the cloud service (or legitimately runs the cloud service) and may also be able to trick the user into adding or deleting files. There are careful proofs that even this powerful attacker cannot read directory names or file contents, nor modify the data without detection.

2.1 A note on "negligible" for non-cryptographers

A function f (of some parameter k) is *negligible* if for all polynomials p, for sufficiently large k,

$$f(k) < 1/p(k)$$

It helps to think of k as the bit-length of the key. If an attacker is trying to guess a key of bit-length k, there are 2^k different possibilities. The probability

that one guess is successful is $1/2^k$. Even if the attacker makes $q(k)$ guesses (where q is a polynomial), the success probability is only $q(k)/2^k$, which is negligible—it becomes exponentially small as k is made large enough.

The ScramFS security analysis carefully quantifies the success of attacks against integrity and privacy, which amount to trying to guess the symmetric encryption key. These are shown to be negligible functions of the key length. In other words, it shows that breaking ScramFS's secrecy and integrity is "not much easier" than guessing the AES key.

Strictly speaking, a number (such as $1/2^{128}$) cannot be negligible, though it is sometimes described as such in the ScramFS security analysis and other technical cryptography writing. This should be interpreted to mean that this value, as a function of the key length, is negligible, and that we happen to have chosen a key length of 128 bits.

3 Technical comments on the design and proofs

It is always possible for the attacker to roll the system back to a prior state—this is an inevitable consequence of the decision not to store state in the client. For example, the attacker can always replace a file with a prior version of the same file—it can replace the entire encrypted filesystem with an earlier version. This is a reasonable decision and is carefully described in the documentation. However, it is important to make sure that the user's mental model of a "prior state" matches that of ScramFS—many subtle security problems are actually the result of slight inconsistencies between the user's model of the system and the assumptions of the system designers. For example, suppose that the user writes a file called "TaxReturn.xls" into the same directory at tax time every year, then deletes it after it has been sent. She might think of this as a new and different file each year, though to ScramFS it's simply the same file and may be rolled back to a prior version. This is not a bug—it is simply an illustration of the importance of making users understand the assumptions of the system.

I understand that the complexity of the key generation function is related to a future set of functionalities necessary for multi-user access to one file system. I suggest that this be the subject of a future, separate, review when those requirements are more clearly defined.

A Minor errors and typos

Note: I am told that these minor errors have been fixed based on an earlier version of this report.

- SIV encryption isn't defined (or rather "SIV encryption" is defined as a particular scheme, but only in the design outline, not the security analysis, and nowhere is the acronym SIV explained).
- In the list of constant values at the start of Section 5 (Cryptography), *Con.Kdircuse.dircMS* = *Con.Kdirluse.IntMS* and *Con.Kdircuse.dircLS* = *Con.Kdirluse.IntLS*. I assume this wasn't intended, because it would seem to produce equal values for $K_{dir.l}$ and $K_{dir.c}$. Although this doesn't seem to be a disaster—see comments on key gen above—perhaps you intended the second pair to be $0^{14}00$ and $0^{14}01$.
- In Fig 2, the second IV has the wrong subscript. Should be $FC, 2$.