Security Analysis of TrueCrypt 7.0a with an Attack on the Keyfile Algorithm

Ubuntu Privacy Remix Team <info@privacy-cd.org>

August 14, 2011

Contents

Preface

We previously have analyzed versions 4.2a, 6.1a and 6.3a of the TrueCrypt program in source code without publishing our results. Now however, for our new analysis of version 7.0a we decided to publish it. We hope that it will help people to form their own sound opinion on the security of TrueCrypt. Moreover, we solicit help in correcting any mistakes that we've made. To this end, we would like to encourage everyone reading this to send criticism or suggestions for further analysis to us.

While preparing the analysis for publication we reassessed our previous results. In doing so we discovered major weaknesses in the TrueCrypt keyfile algorithm. This could even be turned into a successful attack on TrueCrypt keyfiles. We present that attack in section 7. We want to stress that the security of TrueCrypt containers which do not use keyfiles is in no way affected by this weaknesses and the attack.

TrueCrypt is a multi-platform program. Up to now there are versions for Windows, Linux and Mac OS X. Our analysis mainly focuses on the Linux version. The Windows version has been analyzed to a lesser extent, the Mac OS X version not at all. In large parts the code basis is the same for all operating systems on which TrueCrypt runs. On the other hand there is some special code for each of these operating systems. This is even reflected in slightly diverging behavior of the program on different operating systems here and there.

In the source code of TrueCrypt 7.0a there are, moreover, folders for the operating systems FreeBSD and Solaris. Apparently the source code in these folders hasn't reached a point where a program could be built and distributed from it. Therefore, we completely neglected them.

The report at hand explains the results of our analysis. It is organized as follows: Section 1 lists some data of the analyzed program. Section 2 contains remarks on binary TrueCrypt packages. Section 3 deals with compiling TrueCrypt from the sources. Section 4 explains the methodology of our analysis. In section 5 we describe our program `tcanalyzer` which has been written for this analysis. Section 6 contains our findings in detail except for the attack on keyfiles to which section 7 is devoted. Finally section 8 presents our conclusions. The rational for the conclusions in section 8 is mainly presented in section 6. In sections 6 and 7 some elaborated technical or mathematical facts have been documented in the footnotes. Readers who don't have the special skills to understand them may safely ignore them.

1. Data of the Program

Website:	http://www.truecrypt.org/
Analyzed version:	`TrueCrypt 7.0a`
Analyzed source code (Unix):	`TrueCrypt 7.0a Source.tar.gz`
MD5 fingerprint:	`0a61616bc5c5ad90e876b4014c004ac9`
SHA1 fingerprint:	`42be0f333e6791e7a122b3e1183e014cd3860198`
Alternative source code (Windows):	`TrueCrypt 7.0a Source.zip`
MD5 fingerprint:	`752479c674bc18d6bcf55d056560f0a7`
SHA1 fingerprint:	`8f9bf2ae13461fb3bfb4d1f7acb76c7c1c7ed29d`

2. Remarks on Binary Packages of TrueCrypt 7.0a

As for Windows binary packages of TrueCrypt 7.0a are distributed as executable installation programs for Linux too although they are packed once more in a .tar.gz archive. After unpacking an installation program its execution offers the options of either directly installing TrueCrypt 7.0a or extracting another .tar.gz archive which would be put into the `/tmp` folder.

Despite the fact that executable installation programs are uncommon on Linux anyway this behavior is a further move away from the usual Linux package management of software. Up to TrueCrypt 6.3a the installation program extracted or installed a .deb or .rpm package. Thus it could be uninstalled or updated with the package management of the Linux distribution which is no longer possible with TrueCrypt 7.0 or 7.0a. As a compensation TrueCrypt 7.0 or 7.0a now installs the script `/usr/bin/truecrypt-uninstall.sh` by which it could be uninstalled again.

3. Compiling TrueCrypt 7.0a from Sources

Compiling TrueCrypt 7.0a on Linux

We tested compiling TrueCrypt 7.0a on Linux on an Ubuntu 10.04 LTS desktop system. In addition to the packages of the default system installation the following packages are needed for building TrueCrypt

- `g++`
- `nasm`
- `libwxgtk2.8-dev`
- `libfuse-dev`

and all the packages of which they are dependent. Furthermore, as the file `Readme.txt` in the sources tells, the three header files

- `pkcs11.h`
- `pkcs11f.h`
- `pkcs11t.h`

must be downloaded from `ftp://ftp.rsasecurity.com/pub/pkcs-11/v2-20` and copied to `/usr/include`. If you don't want to have them in this standard include directory you may put them somewhere else and define and export the environment variable `PKCS11_INC` with the contents of the complete path of the directory where you put them. Then we called

```
tar xvzf "TrueCrypt 7.0a Source.tar.gz"
cd truecrypt-7.0a-source
make
```

The compiler run went through without any errors or warnings and in the sub-folder `Main` the executable program `truecrypt` was found. Compiled like that the program supports the command line as well as a GUI.

If you want to build a Debian package you additionally need the package `fakeroot` and may do the following:

```
mkdir -p truecrypt_7.0a/DEBIAN
mkdir -p truecrypt_7.0a/usr/share/applications
mkdir -p truecrypt_7.0a/usr/share/pixmaps
mkdir -p truecrypt_7.0a/usr/share/truecrypt/doc
```

```
     mkdir -p truecrypt_7.0a/usr/bin
     echo "Package: truecrypt" > truecrypt_7.0a/DEBIAN/control
     echo "Version: 7.0a" >> truecrypt_7.0a/DEBIAN/control
     echo "Section: utils" >> truecrypt_7.0a/DEBIAN/control
     echo "Priority: optional" >> truecrypt_7.0a/DEBIAN/control
     echo "Architecture: i386" >> truecrypt_7.0a/DEBIAN/control
     echo 'Maintainer: name <email>' >> \
truecrypt_7.0a/DEBIAN/control
     echo "Depends: libwxgtk2.8-0, libfuse2" >> \
truecrypt_7.0a/DEBIAN/control
     echo "Description: Software for on-the-fly-encrypted volumes" >> \
truecrypt_7.0a/DEBIAN/control
     cp Main/truecrypt truecrypt_7.0a/usr/bin
     echo "[Desktop Entry]" > \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Encoding=UTF-8" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Name=TrueCrypt" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Exec=/usr/bin/truecrypt" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Icon=truecrypt" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Terminal=false" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     echo "Type=Application" >> \
truecrypt_7.0a/usr/share/applications/truecrypt.desktop
     cp "Release/Setup Files/TrueCrypt User Guide.pdf" \
truecrypt_7.0a/usr/share/truecrypt/doc
     cp License.txt truecrypt_7.0a/usr/share/truecrypt/doc
     cp Resources/Icons/TrueCrypt-48x48.xpm \
truecrypt_7.0a/usr/share/pixmaps/truecrypt.xpm
     fakeroot dpkg-deb --build truecrypt_7.0a
```

The result is a package `truecrypt_7.0a.deb` in your current working directory. The architecture must be `amd64` instead of `i386` if you are building on a 64-bit system. The name and email of the maintainer have to be adapted according to your preferences.

Compiling TrueCrypt 7.0a on Windows

In addition to the Linux build we tested compiling TrueCrypt 7.0a on a Windows XP system. Following the instructions in the file `Readme.txt` in the sources we installed *"Visual Studio 2008 Professional Edition"* in standard configuration. A trial version of this may be downloaded from Microsoft and is valid for 90 days. We also tried building TrueCrypt 7.0a with *"Visual Studio 2008 Express Edition with SP1"* but it missed the folder `atlmfc` with libraries and header files and therefore didn't work.

Next we installed *" Microsoft Windows SDK 7.0 for Windows 7 and .NET Framework 3.5 SP1"*. It is important to install this after the installation of *Visual Studio* because otherwise the variable `WindowsSdkDir` has the wrong value `C:\Programme\Microsoft SDKs\Windows\v6.0A` in the end and the wrong SDK would be used.

Furthermore, we installed *"Microsoft Visual C++ 1.52c"* and set the environment variable `MSVC16_ROOT` to the value C:\MSVC which was the directory where it was installed. Finally we installed *"Microsoft Windows Driver Kit 7.1.0"* in standard configuration and *"NASM 2.08"* for Windows. We also added the path to the program `nasm.exe` to our `PATH` variable.

We put the three above-mentioned PKCS-11 header files in a newly created directory. And we assigned this path as value to a new environment variable `PKCS11_INC`. TrueCrypt 7.0a refuses to build if the project file is placed in a folder the path of which contains spaces. Therefore, we have chosen a path without spaces where we unpacked the archive "`TrueCrypt 7.0a Source.zip`".

After these preparations we loaded `TrueCrypt.sln` in *"Visual Studio 2008"* as project, selected *"All"* as sub-projects to be built and started building TrueCrypt with `F7`. It went through without errors and in the sub-folder `Release\Setup Files` of our project folder the executable `True-Crypt Setup.exe` has been created. This, however, is not the desired TrueCrypt installer. The installer must be created by calling

```
    TrueCrypt Setup.exe /p
```

within this sub-folder. This call creates `TrueCrypt Setup 7.0a.exe` which finally is the desired installer.

4. Methodology of Analysis

We carefully read the source code of the Unix source code archive. The parts common to all operating systems and the special part for Linux were completely read. It helped to create diffs to the source code of version 6.3a which we analyzed in 2010 in the same way. Thereby we could focus our attention on code that has changed substantially.

For the XTS mode–which is used by TrueCrypt 5, 6 and 7–we have studied the mathematical theory in relevant cryptographic publications as we did in 2006 for the LRW mode used by TrueCrypt 4.2a. The XTS mode is state of the art in cryptography for disk or volume encryption and replaces the LRW mode in this respect.

We then have written the program `tcanalyzer` which analyzes headers of TrueCrypt containers. It makes use of the libraries of *ScramDisk for Linux* (see http://sd4l.sourceforge.net/) which is an independent development which could read and also create TrueCrypt containers. For this purpose we downloaded http://sourceforge.net/projects/sd4l/files/sd4l/2.1/ScramDisk_2.1-0.tar.gz and linked our program against the libraries built from this source code. Our `tcanalyzer` program is put under the GNU General Public License version 3 and published together with this analysis in source code.

We created test containers as well with the binaries we built ourselves on Linux and Windows as with the binary packages downloaded from the TrueCrypt website. All ciphers and all digests have been chosen at least once for a test container. Hidden containers within outer containers were also among the test containers created. They were created sometimes directly together with the outer container and sometimes within a previously created outer container. We analyzed all headers of all theses test containers with our `tcanalyzer` program. In the end we were convinced that there were no mistakes or back doors in the encryption or its header format.

On Windows TrueCrypt 5, 6 and 7 can encrypt the entire operating system. We only once have created such a system encryption with TrueCrypt 7.0a and analyzed an image of the encrypted disk with `tcanalyzer`. In this case we also convinced ourselves that the format of system encryption differing from the usual TrueCrypt encryption format is correct. We haven't analyzed or even read the TrueCrypt boot code that decrypts such a system encryption in the boot process when the computer starts.

5. The program tcanalyzer

The `tcanalyzer` program analyzes one of the up to four headers of a TrueCrypt container and displays the information contained in that header. It makes use of the *ScramDisk for Linux* code (see http://sd4l.sourceforge.net/) as a library. Which header is analyzed is controlled by command line arguments of `tcanalyzer`. Output is either written to the terminal or to a file given on the command line. `tcanalyzer` recognizes the format of TrueCrypt versions 4.1 to 7.0.

This program must only be used on test containers. Otherwise the security of your data might be compromised as the key to the container will be displayed.

The `tcanalyzer` program has the following options to be given on the command line:

- **-?, --help :** print a help message and exit.
- **-b, --backup:** analyze the backup header.
- **-h, --hidden:** analyze the hidden volume.
- **-s, --sysdisk:** assume the container is a system disk.
- **-f, --full:** print the full header (64 kilobytes for versions > 5), without this option only 512 bytes are printed.
- **-o, --out *file*:** write output to the given file, without this option output is written to the terminal.
- **-p, --passphrase *password*:** use the given password for analyzing, without this you will be queried for the password.
- **-c, --container *container*:** analyze the given container.

At typical output of tcanalyzer looks as follows:

```
Enter passphrase:
:> ****
Analysis of normal header for TrueCrypt container /tmp/test-70a-aes.tc
================================================================================
TrueCrypt version : 7
Encryption mode   : XTS
Digest            : ripemd160
Cipher            : aes
Key size          : 32 bytes
================================================================================
    0: cc c3 d2 be 41 46 ba 09 b3 09 cf d9 b1 a1 c4 d8 | salt
   16: 41 fa 47 aa e1 50 63 ad 48 46 44 19 20 ac 7c 72 | salt
   32: 5e f4 00 de 5b 6f 0f 52 90 af 82 e4 f4 3c 51 61 | salt
   48: 29 78 b8 21 87 4c 79 02 02 d2 c7 b0 02 82 48 13 | salt
   64: 54 52 55 45 00 05 07 00 ce e4 b4 c6 00 00 00 00 | 64: 'TRUE', 68: header
version 5, 70: program version 7.0, 72: CRC32(bytes 256-511)
   80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 92: hidden volume
size 0
   96: 00 00 00 00 00 00 00 00 00 3c 00 00 00 00 00 00 | 100: volume
size 3932160, 108: master encryption offset 131072
  112: 00 02 00 00 00 00 00 00 00 3c 00 00 00 00 00 00 | 116: master encryption
size 3932160, 124: flags 0
  128: 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 | 128: sector size 512
  144: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  176: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  192: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  208: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  224: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  240: 00 00 00 00 00 00 00 00 00 00 00 00 b3 5b ea 86 | 252: CRC32(bytes
64-251)
  256: 59 e5 89 3a fd 4e 41 ca 19 89 2f 5f 74 ee 10 48 | master key
  272: 96 d4 e9 22 9d b0 da d8 41 7a 52 2d 40 65 fd 9f | master key
  288: 8a d9 1a a1 66 43 24 5e 71 04 4a 00 52 2c d7 a4 | XTS key
  304: 12 be 55 e9 89 ca 29 11 ed ec 0c 7f ac 62 cc af | XTS key
  320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  336: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  352: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  368: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  384: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  416: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  432: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  448: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  464: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
  496: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
```

After the information on the TrueCrypt version which created the container, the encryption mode, hash and encryption algorithm and key size the decrypted bytes of the header are displayed with 16 bytes per line. Each line is followed by a token "|" separating some explanation on the meaning of the data in that line.

6. Findings of Analysis

The TrueCrypt License

TrueCrypt makes use of cryptographic code from five different sources: from the TrueCrypt Foundation itself, from Paul Le Roux, Brian Gladman, Mark Adler and Eric Young. Therefore, the TrueCrypt license–in its current version 3.0–contains three further sections with regulations from Paul Le Roux, Brian Gladman and Mark Adler besides the own regulations of the TrueCrypt Foundation. The copyright of Eric Young is mentioned but not covered by a section in the license file. Thus far the situation has historical reasons as TrueCrypt originally arose as a derivative of "Encryption for the Masses" (E4M). But the own regulations of the TrueCrypt foundation are also incompatible with the GNU General Public License (GPL). This situation complicates the legal status of the TrueCrypt sources. A continuation by others in case the TrueCrypt Foundation ceases to maintain TrueCrypt or the development of a spinoff product would be very problematic not to say illicit. Also the major Linux distributors Debian, Ubuntu, Fedora, openSUSE and Gentoo do not consider TrueCrypt as an open source product and therefore refuse to provide TrueCrypt packages (see http://en.wikipedia.org/wiki/TrueCrypt#Licensing).

Website and Documentation of TrueCrypt

The TrueCrypt website gives a somewhat uncommunicative impression. Only the last version 7.0a and very few past versions–for Linux only version 6.3a–can be downloaded from the TrueCrypt website. Bug reports are not wanted in the TrueCrypt forum. There is a form for bug reports on the website but postings on this form were never published. Also the parts of the forum dealing with problems are only readable by logged in users. The TrueCrypt developers never reacted on suggestions or hints we made no matter whether posted in the forum or sent by email to addresses published on the website. This, however, changed when we posted the current attack on keyfiles.

We got an immediate response on this bug report. We discuss this response together with the description of our attack in section 7.

On the other hand the website presents a good documentation of the program. It consists of a manual explaining every feature of the program and also of a detailed technical part. The latter even enlightens the cryptographic procedures and the container format and aided our analysis. What we definitely miss is a changelog outlining the code changes from one version to the next with a short reasoning what purpose they serve. This lack seriously hampers the code analysis of new versions.

Cryptographic Algorithms of TrueCrypt

TrueCrypt 7.0a offers three different hash and eight different encryption algorithms to be selected on container creation. The selected hash algorithm is used to derive the key which encrypts the header of the container from the password supplied by the user. The following hash algorithms are offered for choice:

- Ripemd-160,
- SHA-512 and
- Whirlpool.

The encryption algorithms are composed of three ciphers which constitute the encryption algorithm either singly or in succession. These encryption algorithms are[1]:

- AES,
- Serpent,
- Twofish,
- AES-Twofish,
- AES-Twofish-Serpent,
- Serpent-AES,
- Serpent-Twofish-AES and
- Twofish-Serpent.

Further algorithms used by past versions of TrueCrypt are still present in TrueCrypt 7.0a but are only used when opening old containers created with those deprecated algorithms. The correctness of implementation of all those algorithms has been proven with our program `tcanalyzer` with which we successfully decrypted what has been encrypted by TrueCrypt 7.0a. Moreover, the implementation of the encryption algorithms in the Linux kernel is fully compatible with this implementation as TrueCrypt delegates the encryption and decryption of the volume on Linux to the kernel.

A combination of several ciphers enhances the security of the container which is enciphered by it. This is so because a weakness or even beyond that a break of one of those ciphers which might be discovered sometime in the future would not threaten the security of the container as it is still protected by another unbroken cipher. On the other hand the selection of a combination of several ciphers slows down encryption and decryption. On aged hardware this might cause an unbearable system load. For this reasons we consider Serpent-AES a good choice for containers to be used

1 All three ciphers were finalists in the world wide competition by which Rijndael was selected by NIST (National Institute of Standards and Technology of the USA) for Advanced Encryption Standard (AES) on October 2, 2000 (see http://csrc.nist.gov/archive/aes/index.html). All three ciphers have been designed by teams of well-known cryptographers and scrutinized publicly and intensively in the process of that competition. No security holes have been found in these ciphers. For combinations of these ciphers the order specified by TrueCrypt is that of decryption. On encryption and on initialization the ciphers are used in the opposite order.

on hardware which is fast enough. On hardware which isn't fast enough we would recommend AES alone as it is an international standard.

The choice of the hash algorithm is not that important from a security point of view. We would recommend SHA-512 as it too is an international standard. Moreover the length of the hash value of SHA-512 is 64 bytes (512 bits) which is considerably more than the 20 byte (160 bits) hash length of Ripemd-160. Whirlpool has the same hash length as SHA-512.

TrueCrpyt likewise restricts the length of the password to a maximum of 64 bytes. This isn't a problem as longer passwords not really make sense. For longer passwords the fixed length of the key would be the limiting factor for the security. If you avoid sensible strings or words of a spoken language in the password[2] 30 or may be even 20 characters suffice to impede even major intelligence agencies from breaking the container. With a character set of 62 characters (lower-case and upper-case letters and digits) 20 characters amount to 119 bit of entropy[3]. With the diceware method[4] for creating passwords nine diced words would amount to 116 bit of entropy. On average they would have 38.1 characters with the English word list.

In order to counter the weakness of short passwords a little TrueCrypt repeats the application of the hash algorithm 1000 or 2000 times respectively[5]. For attacks the effect is the same as if the password had 10 or 11 bits more entropy. With the speed of desktop computers common today the iteration count could be much higher for example 100000. Introducing such a higher iteration count in a new version of TrueCrypt would break forward compatibility which is, however, broken by each major new release of TrueCrypt anyway.

Cryptographic Modes used by TrueCrypt

The encryption algorithms are operating on blocks of 16 bytes (128 bits). The deprecated algorithms of past TrueCrypt versions are operating on blocks of 8 bytes (64 bits). They are only used by TrueCrypt 7.0a when opening old containers created with those algorithms. So as not to encrypt blocks with identical contents to identical blocks with the encrypted container the encryption is modified slightly from one block to the other. This is called the encryption mode. Up to version 4.0 TrueCrypt applied the Cipher Block Chaining mode (CBC) for this purpose. This mode, however, has considerable weaknesses if applied in the context of a volume encryption. By this weaknesses an active attacker, who is able cause the legitimate user of the container to store some special data in the container, could later draw conclusions on other data within the container.

In order to close that security breach TrueCrypt replaced the CBC mode with the LRW mode in version 4.1. The CBC mode was further supported for backward compatibility but new containers were only created with the LRW mode. The LRW mode is named after its inventers Liskov, Rivest and Wagner[6]. In this mode a second key of the same length as the block length is multiplied with a block counter in a Galois field and the result of this mathematical operation is added as well to the

---

2 In order to emphasize the necessity to avoid sensible words–or at least to have more than one word–in this context PGP and GnuPG use the term “passphrase” instead of “password”. We, however, stick here with the terminology used by TrueCrypt.

3 The entropy in bits in this context is a measure for the strength of the password. The number of distinct passwords an attacker must try until he certainly hits the correct password will be $2^{entropy}$ which means that every additional bit of entropy doubles the effort for the attacker.

4 See the diceware home page http://world.std.com/~reinhold/diceware.html for an explanation of this. You may also download the diceware word list from that page.

5 To be precise we note that the hash algorithm isn't applied in its simple form to the password. Rather the keyed-hash message authentication code (HMAC) is applied to the password as key and a 64 byte salt value stored at the beginning of the container. This computation is iterated 2000 times if the hash algorithm is Ripemd-160 or 1000 times for the hash algorithms SHA-512 and Whirlpool. For system encryption the iteration count is 1000 also for Ripemd-160.

6 See Moses Liskov, Ronald L. Rivest and David Wagner: *“Tweakable Block Ciphers”*, CRYPTO 2002.

plain text before encryption of the block as to the cipher text after that encryption. We've checked the mathematical correctness of those operation and the representations[7] of the Galois fields used for block lengths of 64 and 128 bit. Also the coding of this has been checked and is correct. The correctness is further confirmed by the compatibility of this code with independent implementations of this in the Linux kernel and in *ScramDisk for Linux*.

In version 5.0 TrueCrypt introduced the XTS mode as a replacement for the LRW mode[8]. The XTS mode is a slight modification of the LRW mode designed to compensate for a theoretical little weakness of the LRW mode. The smaller of the Galois fields is no longer used with the XTS mode, the other one remains the same as before. The XTS mode introduces encryption units of 512 bytes. Up to version 6.3a these encryption units were identical with the sectors. Versions 7.0 and 7.0a allow for larger sectors but the size of the encryption units is unchanged by that. The sector size is stored in the TrueCrypt volume header but it does not modify the encryption. For each encryption unit the encryption is initialized by an encryption of the number of that unit with a second XTS key then the Galois operations known from the LRW mode are used for the blocks within the encryption unit.

The XTS mode used by TrueCrypt is an up-to-date and mathematically provable secure encryption mode provided that the underlying encryption algorithm is secure. Due to the weaknesses in the CBC mode and in the LRW mode we strongly recommend not to use containers with these modes i.e. containers created with a TrueCrypt version before 5.0.

After the header of a container has been read TrueCrypt 7.0a delegates encryption and decryption of the volume on Linux to the Linux kernel. This is so for containers encrypted with the XTS mode and also for containers encrypted with the LRW mode if the block length of the cipher is 16 bytes (128 bits). So this concerns all containers created with versions 5, 6 and 7 and also some containers created with versions between 4.1 and 4.3a. To this end TrueCrypt hands over the encryption algorithm, the encryption mode, the offset of the encrypted area within the container and the master key for encryption and for the mode operation in hexadecimal form to the Linux kernel by virtue of the command `dmsetup`.

On the one hand this gives us an independent check for the correctness of the volume encryption as it is confirmed by a successful decryption in the Linux kernel. There is a huge number of developers working on the Linux kernel. Some of them must have scrutinized the cryptographic code in the kernel.

On the other hand the use of `dmsetup` opens an attack window. An attacker might simply read the master key from the output of the command

```
sudo dmsetup table --showkeys
```

However, since this command needs root privileges, this is not an objection against the security of TrueCrypt 7.0a. There are numerous other possibilities for an attacker with root permissions on a system where the container is opened to compromise the security of that container. The lesson from this is that TrueCrypt has to be used only in a secure environment. Otherwise TrueCrypt can't protect your data just as any other encryption program can't. Ubuntu privacy remix was developed precisely to provide such a secure environment.

---

7 These representations are based on the polynomials $x^{64}+x^4+x^3+x+1$ and $x^{128}+x^7+x^2+x+1$ for the Galois fields $GF(2^{64})$ and $GF(2^{128})$ respectively. They are indeed primitive over $GF(2)$ which is the necessary and sufficient condition for the mathematical correctness of those representations. The first polynomial is only relevant for the old algorithms operating on 64 bit blocks. In the XTS mode it is no longer used since this mode only operates in conjunction with the new algorithms.

8 Slightly simplifying the picture, the same Galois multiplication for 16 byte blocks is used in the XTS mode as in the LRW mode. In each 512 byte unit, however, it is newly initialized by an encryption of the number of the unit with the XTS key.

TrueCrypt Volume and Hidden Volumes

TrueCrypt does create encrypted containers as a file or directly as a partition on a hard disk or an entire drive. The latter option has the advantage that no disk space is wasted for the file system outside the container. The former option allows an easy backup mechanism by simply copying the container without opening it. On Ubuntu privacy remix the semi-automatic backup mechanism is bound by the name `backup.tc` to a container file as well.

A further option of TrueCrypt is to create a hidden volume within an outer TrueCrypt container. This is possible as well for container files as for encrypted partitions or entire encrypted drives. Since version 6.0 a second header of 64 kilobytes for the hidden container is located just after the header of the outer container. If no hidden container has been created within the outer container this second header is filled with random data. The header for the hidden container contains the information where in the outer container the hidden container starts. Files within the outer container are located before the start of the hidden container. Which container is opened by TrueCrypt–the outer or the hidden one–just depends on the password supplied. From this it is clear that the outer and the hidden container can't have the same password.

We analyzed the code for creation of hidden containers. The security of the encryption for hidden containers is identical with that of the encryption of normal TrueCrypt containers. Moreover, as far as the encryption isn't broken, there is no way to tell whether there is a hidden container within a given TrueCrypt container. This is so even if the password for the outer container has been handed over to the attacker. This is called “plausible deniability”. In order to achieve this plausible deniability not only the second header of a normal container but also the volume is filled with random data on creation of the outer container. As a good encryption is indistinguishable from random data neither the second header nor the volume data could be used to tell whether there is a hidden volume with the outer volume.

The TrueCrypt documentation describes the purpose of a hidden container as a means to solve a situation where you are forced by someone to reveal the password to an encrypted volume, for example due to extortion. Whether this works in such situations is open to question. The hidden container might however serve its purpose in a legal situation like in Great Britain where an accused in court is committed to hand over his keys and passwords to the court. How the courts are ruling on the conjecture that there is a hidden container within the outer container remains to be seen.

The Random Number Generator of TrueCrypt

As any other good encryption program TrueCrypt needs random numbers. In the case of TrueCrypt these are 64 bytes of *“salt”* used to derive the key for the header encryption from the password. Then it needs between 32 and 96 bytes for the master key by which the volume is encrypted. Another 32 to 96 bytes are needed for the XTS key used in the XTS encryption mode. The size of these keys depends on the number of ciphers of the selected encryption algorithm. Besides this TrueCrypt also fills the volume with random data as is explained in the previous section. To create all these random data is the task of the random number generator which TrueCrypt implements.

The random number generator of TrueCrypt is based on a paper written by Peter Gutmann in 1998[9]. It makes use of mouse positions and times of events like mouse clicks or keyboard entries. These data are practically unpredictable. On a Linux system random values from the pseudo devices `/dev/random` and `/dev/urandom` are added to these data. To date there are no known attacks against this random number generator. But a paper by Kelsey, Schneier, Wagner and Hall[10] where similar though simpler pseudo-random number generators were analyzed evinces that such

9 See Peter Gutmann, *“Software Generation of Practically Strong Random Numbers”*, Usenix Security Symposium 1998

10 John Kelsey, Bruce Schneier, David Wagner and Chris Hall, *“Cryptanalytic Attacks on Pseudorandom Number Generators”*, 1998

an attack may be possible if an attacker could actively interfere on the system while random numbers are being generated. An inference from this analysis was the development of *Yarrow* which is definitely a better pseudo-random number generator[11].

An active attacker with root privileges may also eliminate the randomness from the Linux random devices by the commands

```
sudo rm /dev/random /dev/urandom
sudo mknod /dev/random c 1 5
sudo mknod /dev/urandom c 1 5
```

completely[12]. As similarly argued above this is not an objection against the use of these devices by TrueCrypt but minds us that TrueCrypt must only be used in a secure environment.

The implementation of the random number generator in TrueCrypt has the disadvantage that there is no estimate on the amount of real randomness gathered. That's why it doesn't block until the mouse has been sufficiently moved or sufficient other random events happened. This corresponds to the behavior of the device `/dev/urandom` on Linux in contrast to `/dev/random`. By this behavior the user isn't troubled with the need of lengthy mouse movements but then he is less secure. We recommend to move the mouse for at least one minute before clicking the *"Format"* button on container creation. We also recommend to uncheck the check box for *"Show"* in this dialog. Otherwise 13 bytes of the current random pool are permanently displayed. Afterwards even the first 13 bytes of the header and the master key are displayed. As this could be intercepted by an attacker for example with a receiver for the monitor radiation unchecking this check box is a security demand.

The Format of TrueCrypt Volumes

On file based containers as well as on completely encrypted partitions or drives TrueCrypt applies a header with a size of 64 kilobytes in front of the volume. A second header with identical structure follows for a possible hidden container within the outer container. If there is no hidden container this second header is filled with random values. Moreover, there are two backup headers of the same size and structure at the end of the volume. For system encryption the second header for the hidden volume and the two backup headers are ommited. Also in this case the header has only 512 bytes. In TrueCrypt versions before 6.0 there were no backup headers and the header size was only 512 bytes.

The first 64 bytes of a TrueCrypt header are a salt value selected at random which is stored unencrypted. From this salt value and the password the key is derived by which the remainder of the header is encrypted. Owing to these facts a TrueCrypt header just as the following volume can't be distinguished from random values. Since version 4.2a–the first version we analyzed–the format of TrueCrypt headers changed three times with the versions 5.0, 6.0 and 7.0. The following table specifies the format of a TrueCrypt header in version 7.0 and 7.0a.

Offset	***Size***	***Description***
0	64	Salt (unencrypted)
64	4	Constant check value: ASCII string "TRUE" (encrypted)
68	2	Version number (currently 5) of the header format (encrypted)

11 See John Kelsey, Bruce Schneier and Niels Furguson, *"Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator"*, SAC'99, 1999

12 These commands replace the random devices by a copy of `/dev/zero` which always yields zero bytes when queried.

Offset	***Size***	***Description***
70	2	Version number of the program (currently 7.0) required to open the volume (encrypted)
72	4	CRC-32 checksum of the decrypted header bytes 256-511 (encrypted)
76	16	Unused bytes set to zero (encrypted)
92	8	Size of the hidden volume or zero for non-hidden volumes (encrypted)
100	8	Size of the volume (encrypted)
108	8	Byte offset of the start of the area encrypted with the master key (encrypted)
116	8	Size of the area encrypted with the master key (encrypted)
124	4	Flag bits: bit 0 set means system encryption, bit 1 set means non-system in-place-encrypted volume, bits 2-31 unused (encrypted)
128	4	Sector size in bytes (encrypted)
132	120	Unused bytes set to zero (encrypted)
252	4	CRC-32 checksum of the decrypted header bytes 64-251 (encrypted)
256	Variable 64, 128 or 192	Master key and following it the second key for the XTS mode, this is padded with bytes set to zero up to byte 511 (encrypted)
512	65024	Unused, for system encryption this part of the header is omitted. The Linux version of TrueCrypt sets these bytes to zero whereas the Windows version sets them to **random** values. (encrypted)

As remarked in this table the Windows version of TrueCrypt 7.0a deviates from the Linux version in that it fills the last 65024 bytes of the header with random values whereas the Linux version fills this with encrypted zero bytes. From the point of view of a security analysis the behavior of the Windows version is problematic. By an analysis of the decrypted header data it can't be distinguished whether these are indeed random values or a second encryption of the master and XTS key with a back door password. From the analysis of the source code we could preclude that this is a back door. For the readability of the source code this duplication of code which does the same thing in slightly different ways was however a great impediment. It certainly must also hamper the maintainability of the code.

As it can't be ruled out that the published Windows executable of TrueCrypt 7.0a is compiled from a different source code than the code published in "`TrueCrypt 7.0a Source.zip`" we however can't preclude that the binary Windows package uses the header bytes after the key for a back door. The Linux version does not have that problem with these bytes as their decryption to zero proves that they don't hide a duplicate key.

In principle such a duplicate key could also be hidden within the salt value. The 64 salt bytes would be enough to store the master key and the XTS key for an encryption with a single cipher. This could be encrypted with a fixed key known to the vendor of the binary package or possibly to someone who payed the vendor for the back door. Such a back door would be possible also with the binary packages for Linux. If a combination of two or three ciphers has been selected for the container the 64 bytes of salt do not suffice to store the key there but then the salt bytes of the backup header could be used in addition.

You however best protect yourself against such possible back doors by compiling the publicly analyzed source code yourself. We reported SHA1 and MD5 fingerprints of the analyzed source code in section 1 for the purpose that you may check that the source code you downloaded from http://www.truecrypt.org/ is the identical to the source code we have analyzed. You may check this on a Linux system by applying `md5sum` or `sha1sum` to the source code archive.

In case of a system encryption of the Windows system partition there is only one 512 byte header. This header isn't stored in the system partition itself but in the last sector of the drive before the first partition. This usually is sector 62 on a drive with 512 byte sectors which are counted beginning with 0.

7. An Attack on TrueCrypt Keyfiles

The TrueCrypt Keyfile Algorithm

The so called "*keyfiles"* of TrueCrypt are meant to supplement the password or–if the password is empty–to replace it. A homebrew algorithm[13] is applied to the contents of one or several keyfiles modifying the password. One objection against the use of keyfiles is the following: Those keyfiles must be stored somewhere. If an attacker can get access to the container he might as well get access to the keyfile or keyfiles if there are more than one. An exception from this objection may be the storage of a keyfile on a PKCS-11 compliant security token protected by a PIN code. If the attacker gets access to the files which may have been used as keyfiles it's not a great impediment to test all of them even if there are millions of them to be considered. In conjunction with a bad or even empty password the security of the container may soon be broken.

To our knowledge the TrueCrypt keyfile algorithm hasn't been analyzed cryptographically before. In the algorithm intermediate states of processing one byte of the keyfile at a time with a CRC-32 checksum[14] are added modulo $2^8=256$ to subsequent bytes of a 64 byte pool. Every time the last byte of the pool has been reached the process continues with the first byte of the pool. Finally the pool is merged with the password which is extended to 64 bytes with appended zero-bytes. This is done by the exclusive-or (XOR) operation. From this we may conclude that the security supplied by the password isn't weakened by the additional use of keyfiles as flipping any bit of the password does change the outcome. On the other hand the security is also not strengthened in any case as our attack below will prove.

It is well-known that the CRC-32 algorithm doesn't meet cryptographic standards of a hash algorithm. Therefore cryptographic security can't be expected of this keyfile algorithm. In fact, we discovered an attack by which any file can be manipulated so that it has no effect at all when added as a keyfile. For this purpose we only need to append between 1524 and 2804 bytes to the file. There are many applications which don't mind the additional bytes at the end of such a manipulated file and present it's contents just as it was without those bytes. This is especially true for image viewers and image files (for example `.jpg` and `.png` files) and for PDF readers (such as the Adobe Acrobat Reader or Evince) and PDF documents.

The Manipulation of TrueCrypt Keyfiles

Our attack will manipulate an arbitrary file so that it has no effect when added as a keyfile in the creation of a container with TrueCrypt. The container may afterwards be opened with or without the keyfile and with the same password in both cases. Vice versa, any container created without a key-

13 See http://www.truecrypt.org/docs/?s=keyfiles-technical-details for a description of the algorithm. The documentation is however wrong in one respect. It speaks of a hash function whereas in the code only a cyclic redundancy code (CRC-32) is applied.

14 The CRC-32 algorithm used here is that of the standard IEEE802.3 which is also used in Ethernet transmissions.

file may afterwards be opened with this manipulated keyfile and its password. If several different such manipulated keyfiles have been added in the creation of a container some or even all of them may afterwards be omitted when opening the container. An evil attacker could spread many such manipulated files in places where as he hopes his victim will select his keyfiles from. An alternative attack line would be to distribute a Trojan horse which secretly does this manipulation to all files the victim processes with it.

As a demonstration of our attack we have modified the PDF file of this document according to the attack. Likewise we have done with the German version of this document. Opened with a PDF viewer the contents is still displayed as we have written it. As a verification of the attack you may however add one or even both versions of this document as keyfiles to a new TrueCrypt container. The container will afterwards equally open without these keyfiles.

The first step in our attack is to append the CRC-32 checksum of the file to be manipulated at the end of that file. It is a special property of the CRC-32 algorithm that the CRC-32 checksum of this concatenation is zero. This property is well-known and commonly used in the verification of CRC-32 checksums. For the TrueCrypt keyfile algorithm this means that the intermediate state of the CRC-32 algorithm becomes zero after processing the original file and the four appended bytes. For our attack this has the advantage that it can now proceed with an initial situation which is not only simple but always the same.

In developing our attack at first we calculated the 256 sequences of five bytes each which yield a CRC-32 state of zero after the fifth byte if the state was zero before applying that sequence[15]. An arbitrary sequence of five bytes would change 20 bytes of the pool but due to the special nature of these sequences they only change 10 bytes of the pool with only four distinct additions if the CRC-32 state was zero before the application. As the addition values to the first byte for these 256 sequences take all possible byte values we can make the first byte after the current position in the pool to zero by just appending that five byte sequence which has the complement modulo 256 of the pool byte as its first addition value.

After this operation the position in the pool has moved 20 bytes forward. Perhaps if we had reached the end of the pool in between we had continued at the beginning of the pool . So we can repeat that operation. Due to the special nature of our sequences–which change only 10 out of 20 bytes–that byte is never changed again by other such operations launched at different pool positions. After 16 such operations we are back to the original position in the pool and have made 16 of the 64 pool bytes zero. For this we had to append another 80 bytes to the file.

To make further bytes of the pool zero without changing the 16 bytes which now are already zero we formed pairs of our five byte sequences which in total add zero to the first byte after the current position in the pool if both were applied at that same position. We sorted these pairs according to the value they add in total to the second byte after the current position. After removing duplicates in that values 44 such pairs remained. If a pool value is the complement of one of that 44 values from these pairs modulo 256 it can be made zero by appending the first sequence of that pair and after 15 other such operations when we are back to the original pool position appending the second sequence of that pair. If a pool value is not the complement of one of those 44 values we could make it zero by adding two or in rare cases three such values. In practice we had to append between 320 and 480 bytes in this step. After this step 32 of the 64 pool bytes were zero.

In the next step we calculated the triples of five byte sequences which in total add zero to the first and the second byte after the current position in the pool. We sorted them according to the value they add in total to the third byte and then again removed triples which only provided duplicates in that value which leaves us with 19 such triples. In this step we needed between two and four triples to make one of the remaining non-zero bytes of the pool zero by the addition of two to four of those 19 values. In practice we had to append between 480 and 960 bytes to the keyfile in this step. The

15 Viewed as binary polynomials–as it is done in the CRC-32 algorithm–these sequences are just the binary polynomials of degree less than 40 which are multiples of the CRC-32 polynomial

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1 \quad .$$

pool after this step had 48 zero-bytes.

Finally, to make the last 16 bytes of the pool zero, we calculated those quadruples of five byte sequences which in total add zero to the first, second and third byte of the pool after the current position. After sorting them according to the value added in total to the fourth byte and removing duplicates there also remained 19 such quadruples. Again we needed between two and four of them to make one of the 16 remaining non-zero bytes of the pool zero. So after appending another 640 to 1280 bytes to the keyfile the pool becomes completely zero.

As a pool which has all bytes zero doesn't change the password when merged with it by the exclusive-or (XOR) operation, we have reached our goal. Summing up we had to append between 1524 and 2804 bytes to a file to annul its effect as keyfile.

It were three major weaknesses in the TrueCrypt keyfile algorithm which facilitated the attack. The first weakness was the application of the cryptographically insecure CRC-32 algorithm. Its input can easily be crafted to yield any desired output. The second weakness was the linearity of the operations by which the pool bytes were changed, that is the additions. This made the attack a simple application of linear algebra. The third weakness was the local nature of changes to the pool. Every byte of the keyfile only changes four bytes of the pool. This enabled us to do the attack successively making one byte of the pool after the other zero.

If the developers of the TrueCrypt foundation don't want to dispense with keyfiles we would recommend using the keyed-hash message authentication code HMAC based on SHA-512 or Whirlpool with the password as key and the complete keyfile as message. If several keyfiles were selected the results of these HMAC operations for the different keyfiles can be merged with the exclusive-or (XOR) operation. The final result can still be supplied to the key derivation algorithm as it is currently done.

Another remark is in order. One should not feel safe if the selected keyfile has an effect and the container can't be opened without the keyfile afterwards. Our attack could easily be modified so as to do a certain change to the password known to the attacker but not to the victim. As we do not want to facilitate such an attack in real life we do not publish our program for the attack and its source code.

Response to the Attack by the TrueCrypt Developers

We posted a bug report concerning the attack on keyfiles on the TrueCrypt website. The same day the TrueCrypt developers replied:

> *“Hello,*
>
> *First, thank you for reviewing our software–we really welcome and appreciate all independent reviews. It should be noted that when we designed the keyfile processing algorithm, we had been very well aware of the properties of the CRC-32 algorithm that you mentioned in the document.*
>
> *Below is our response to the attack you reported to us:*
>
> *No matter what keyfile processing algorithm is used, if the attacker can modify the content of your keyfile before you create your volume, he can reduce the entropy/strength of the keyfile. Just as you cannot allow an attacker to modify or prepare your password when you create a volume, you cannot allow an attacker to modify or prepare your keyfile before you create your volume, either. Otherwise, if you do that, you cannot reasonably expect to have any security.*
>
> *It is a basic security requirement that cryptographic keys (whether passwords, keyfiles, or master keys) must be secret and unknown to attackers. Your attack violates this requirement and is therefore invalid/bogus.*
>
> *This is valid whether you use CRC-32 or HMAC-SHA-512, whether you use the TrueCrypt keyfile processing algorithm or something different.*

> *Thank you again for taking the time to review our software. We hope that the mistakes will be corrected before the review is published (publishing an invalid attack would be misleading).*
>
> *Sincerely,*
>
> *David*
>
> *PS - Even if the attack was valid (it is not) and a cryptographically secure hash (such as HMAC-SHA-512) was used instead of CRC-32 (your suggested fix), the attacker would still be able to 'crack' your keyfile by a comparatively short brute force attack (it would only be slightly slower, as the attacker would merely have to find the correct keyfile among the thousands or millions of keyfiles he crafted).”*

This response is mistaken. It fails to differentiate between the secrecy of a password or key and the inability of an attacker to modify the password or key. The secrecy of the password and keyfile is indeed a basic prerequisite for the security of a TrueCrypt volume as it is in any other cryptographic application. However, it is quite possible that an attacker is able to modify a keyfile to a certain extent while that keyfile at the same time remains perfectly secret and unknown to him. The point to be considered here is that cryptography has means to provide security even in such a weird situation.

As an example for such a situation consider an attacker who distributes an image editing software. He might add our attack algorithm to his software so that it secretly manipulates any image it processes to have no effect as a TrueCrypt keyfile. Our algorithm is even small enough to run on embedded systems. Therefore, even a digital camera might process the pictures it takes and output manipulated picture files. The pictures would nevertheless show what the user wanted and would remain unknown the the vendor of the camera or the image editing software. So they may be kept secret although they have been modified by an attacker.

In that situation a keyfile algorithm based on HMAC-SHA-512 would provide perfect security while the TrueCrypt keyfile algorithm fails completely. A brute force attack would be impossible as the attacker has not to find the correct keyfile among millions of keyfiles he crafted but among a practically infinite number of possible pictures which might have been taken with his camera or processed with his image editing software.

Another scenario where the reasoning of the TrueCrypt developers does not apply is an off-line security system like Ubuntu privacy remix. It has been especially designed to protect the secrecy of files processed within that system, i.e. it is difficult for an attacker to get access to any file processed within that system. However, it is not that difficult to induce the user to insert some files into that system. Therefore, an encryption program inside that system should especially protect against manipulations like the above attack.

8. Conclusion

TrueCrypt 7.0a is a highly secure program for encrypting containers based on the current state of the art in cryptography. We found no back door or security-related mistake in the published source code except for our attack on keyfiles. If you use this program in a secure environment such as Ubuntu privacy remix you may assume with high certainty that no one can get access to the data stored in your containers as long as they are closed, the passwords are really good and the attacker doesn't apply highly advanced methods below the layer of the operation system, such as BIOS rootkits, hardware keyloggers or video surveillance.

There is a fundamental problem with the analysis of binary packages published on the TrueCrypt website. Without a very expensive “reverse engineering” it can't be proved that they are compiled from the published source code. Since we haven't done such a reverse engineering we can't preclude that there is a back door hidden within those binary packages. As argued in section 6 our `tcanalyzer` program also can't rule out the possibility of a back door in a binary package. Therefore we recommend to compile your binaries yourself from the published source code for not to put blind confidence in the TrueCrypt Foundation. In section 3 we gave some details how this could be done.

From the analysis in sections 6 and 7 the following particular conclusions may be drawn. The rational behind these conclusions is given in the sections mentioned.

- All encryption and hash algorithms are good. If your hardware is fast enough you could enhance the security by selecting a combination of several ciphers such as Serpent-AES, if not the standard AES is a good choice for you. For the hash algorithm we recommend SHA-512.
- Old containers created with TrueCrypt versions before 5.0 should be replaced by new containers created with TrueCrypt 7.0a. Such an old container should be deleted after copying its contents to a new container.
- The use of keyfiles is insecure. They doesn't weaken the security supplied by the password used in conjunction with a keyfile but if a weak or even empty password is used with keyfiles you are no longer secure.
- Hidden volumes are secure and provide you with “plausible deniability”. That means that an attacker can't distinguish whether there is a hidden volume within an outer container or not even if the password for the outer container is revealed to him.
- On creating a container we recommend to uncheck the check box for *“Show”* in the last dialog and to wiggle with the mouse for about a minute before clicking on the the *“Format”* button in that dialog.