Secure Offsite Backup at CERN

Katrine Aam Svendsen

Master's Thesis
Master of Science in Information Security
30 ECTS
Faculty of Computer Science and Media Technology
Gjøvik University College, 2007

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Faculty of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

An incident which results in the loss of data can be devastating for an organization, especially a large research organization. It is therefore important to store the vital data in an offsite location, in case the original data for some reason is rendered unavailable.

CERN – the European Organization for Nuclear Research – is such an organization. To protect themselves against this vulnerability, the IT department, situated at the Meyrin site of the organization, have installed an offsite backup server at the Prevessin site, about 3 kilometers away. A live copy of the vital data is backed up to this server regularly.

Some of the groups and services at CERN handle confidential information, such as information about personnel and salaries, and it was therefore desirable to encrypt the data stored on the offsite backup server. This thesis presents the development and implementation of this system, by the use of existing components, as well as policies and procedures needed to administrate and manage the system.

The system incorporates AFS, with Kerberos authentication, which is already in use at CERN. For encryption, TrueCrypt is installed on the backup server. This provides on-the-fly encryption of the transferred data in encrypted volumes mounted as virtual disks. The system ensures that the data is transferred securely, that one user does not have access to any other user's data, and that a physical attack on the server, e.g. theft, do not give an adversary any access to data.

The thesis presents the system design, designed with basis in the initial requirements to the system, and a version adapted to the situation at CERN at the time of implementation. It is shown that the system is adaptable to several different situations.

Sammendrag

En hendelse som resulterer i tap av data kan være ødeleggende for en organisasjon, og dette gjelder også store forskningsinstitusjoner. Det er derfor viktig å ha en offsite backup løsning, på et område som er fysisk adskilt fra stedet der dataene i utgangspunktet oppbevares, der man kan lagre vitale, driftsnødvendige data i tilfelle de opprinnelige dataene av en eller annen grunn blir utilgjengelige.

CERN, den europeiske organisasjonen for kjernefysisk forskning, er en slik organisasjon. For å sikre seg mot denne sårbarheten har IT-avdelingen, som befinner seg i den delen av CERN som ligger i Meyrin, Sveits, installert en offsite backupserver i den delen av CERN som ligger i Prevessin, Frankrike, ca. 3 kilometer unna. En gjeldende kopi av de driftskritiske dataene kopieres til denne serveren regelmessig.

Noen av gruppene og tjenestene ved CERN håndterer konfidensiell informasjon, slik som personell- og lønnsopplysninger, og det er derfor ønskelig å kryptere dataene som lagres på den eksterne backupserveren. Denne rapporten presenterer utviklingen og implementeringen av et sikkert system for å beskytte disse dataene, ved å bruke eksisterende løsninger, i tillegg til nødvendige policyer og prosedyrer som trengs for å administrere systemet.

Systemet benytter AFS, med Kerberos autentisering, som allerede benyttes på CERN. For kryptering er TrueCrypt installert på backupserveren. Dette muliggjør "on-the-fly" kryptering av dataene som overføres og lagres på krypterte volumer som monteres som virtuelle disker. Systemet sikrer at dataene overføres sikkert, at en bruker ikke har tilgang til en annen brukers data, og at et fysisk angrep på serveren, for eksempel tyveri, ikke vil gi angriperen tilgang til dataene.

Rapporten presenterer systemdesignet, som er utviklet på grunnlag av de opprinnelige systemkravene, i tillegg til en versjon som er tilpasset situasjonen ved CERN i skrivende stund. Det er vist at systemet kan tilpasses flere forksjellige situasjoner.

Preface

I would like to thank my supervisor, Chik How Tan, for much help when formulating the project description, feedback on the proposed solution and help with the final report. Also, my colleagues in the IT-DES-SIS section at CERN and the rest of the IT-DES group, as well as personnel responsible for the information security and disaster recovery planning at CERN, have helped in the process of designing and developing the solutions presented in this thesis.

Finally, many thanks to my fellow students Roger and Line, for valuable help, feedback and input on my project and thesis.

Katrine Aam Svendsen, 31st October 2007

Contents

List of Figures

1 Introduction

1.1 Topic

This thesis concerns the implementation of encryption on an offsite backup server at CERN in Geneva, Switzerland. Areas covered are to choose and implement the encryption scheme, design policies and procedures for key generation, -distribution and -handling as well as other administrative tasks, develop a user interface and evaluate the overall security of the implemented system.

1.2 CERN

CERN – The European Organization for Nuclear Research – is the world's largest particle physics centre. It was founded in 1954, as one of Europe's first joint ventures, and do now have 20 member states. At the moment, just under 3000 people are employed by CERN, this being everything from physicists to secretaries. In addition to this, there are about 6500 visiting scientists who come to CERN to do their research.

The laboratory and its physicists have received the physics Nobel price several times. The first time was in 1976, for the discovery of a new kind of elementary particle, and two CERN physicists got the price in 1984 for their contribution in the discovery of two field particles. Another researcher at the laboratory got the price in 1988, and the last Nobel price, so far, received by a CERN physicist was is 1992, for his work on particle detectors [1].

Currently, the main focus at CERN is on completing and starting the LHC, the Large Hadron Collider. This is a particle accelerator that will accelerate particles almost to the speed of light. Different detectors are built to make the particles visible. The goal of the accelerator is to recreate the environment as it was at the origin of our universe, and answer such questions as "why do elementary particles have mass?" and "why is their masses different?" [2].

CERN is located on the border between Switzerland and France, with its main site (Meyrin site) in Switzerland, near Geneva. This is where the main computer centre is located. There are also a site in France (Prevessin site), where, among other things, the control centre to the accelerators is located.

The computer facilities at CERN hold a lot of information which is fundamental for the day-to-day run of the organization, such as personnel information and databases holding information about different experiments. This is backed up in a central network backup system (IBM's Tivoli Storage Manager).

1.3 Problem Description

An offsite backup server, placed at the CERN site in Prevessin, France, is available for the users of the CERN network to store data that is vital for the operation of the institute. The main function of this backup service is to serve as a part of a disaster recovery plan, so that the operation of the organization may proceed as normally as possible if a disaster should occur which results in corruption of the main computer facilities.

Several different types of data may be stored on the backup servers, and it is desirable to keep this data confidential. A good way to ensure this confidentiality is to encrypt the data. However, the information should be available for the users to retrieve at any time, and it should not be too cumbersome to carry out the backups.

When implementing encryption, a scheme for how to distribute and manage keys is a crucial part of the system. The design of this scheme, with guidelines and policies, is an important part of the implementation itself.

Following the implementation of the encryption itself and the key management policies, it is desirable to have a user interface that makes the utilization of the backup server, with the encryption, easy for the users. This apply to both the users administrating the service – creating new encrypted spaces, generating and managing keys etc. – and those using the service to backup their critical data.

Status of the system at the start of the project

The backup server is a computer running Scientific Linux CERN 4.4 (SLC4), situated at the Prevessin site of CERN, in France, approximately 3 km from the main computing central. It is mostly intended to hold remote live backups to use in case of a disaster, or in other situations where the data stored in the main computing center and on regular local backup systems have been destroyed.

At the start of this project, the system is in use, but there is no encryption of the data stored. Users can log on to the machine via SSH[1] to manage the files already backed up, and on Linux machines, files may be backed up using standard Linux commands such as rsync[2] (via SSH) or scp[3]. It is desired that this kind of functionality is preserved after the encryption is implemented. Also, it is desired that the solution can be used from both Linux and Microsoft Windows machines, as CERN use both operating systems.

1.4 Delimitations

This thesis will not consider the overall network security of CERN, but simply the process of securing the data stored on the backup machine and the administrative procedures connected to this process.

Also, it will not be attempted to produce any full information security policy for CERN. The policy presented in chapter 7 is specific for the system presented in this thesis.

1.5 Research questions

1. Is it possible to develop a transparent and secure offsite backup solution within the current systems at CERN?
   - How is the encryption best implemented?
   - How should key management policies and guidelines, and a policy regarding disaster recovery be designed?
   - Is it possible to create a user interface which makes the solution transparent to the user?
2. If so, is the security satisfactory with regards to the requirements to the system?

[1]RFC4251 – The Secure Shell (SSH) Protocol Architecture [3]
[2]rsync is a tool for transferring files, that enables incremental backups. More information in [4, 5]
[3]Secure Copy – file transfer protocol used to copy files safely between computers in a network, via SSH. [6]

2 Choice of method

This project consisted of three phases: a literature study, a development phase consisting of the creation of a system design, installation of TrueCrypt and development of scripts, and design of policies and procedures. This chapter will give a brief description of the different phases.

The project is carried through using an iterative and incremental approach. An iterative process is, according to Robey et al. [7], "designed to include repetition so that work completed in an earlier cycle can be refined and corrected." Further, they assert that an iterative-incremental process is based "on the assumption that it is possible to identify and meet system needs more accurately by continuously revisiting requirements with users." The process of the project is illustrated in figure 1, and how the iterative-incremental approach was carried through in this project is described in chapter 6.

The first period of this project was dedicated to the study of related literature. The main focus was on existing storage solutions, and different kinds of encryption file systems. Information about other ways to solve the problem of this project was also explored. After exploring the state of the art in the technical field of the project, it was also important to obtain information about the writing of policies, key management and disaster recovery planning.

Different sources were examined in this phase, mostly articles available from the major databases, like IEEExplore[1] and ACM[2], and from different books relating to the subject. The outcome of this phase is presented in chapter 3.

During the course of the literature study, it was also important to obtain an accurate knowledge of the situation and current system at CERN, to get a good basis before the development phase. The outcome of this survey is presented in chapter 4.

Based on this knowledge, the development and implementation phase was commenced, starting with the creation of the system design and key management scheme. An important part of this phase was to test possible tools to use in the system design, and make a decision about which to use. This also included some discussions with the relevant people at CERN. After the choice was made, the chosen tool was installed on the backup server and some time was used to get to know it and do some initial tests.

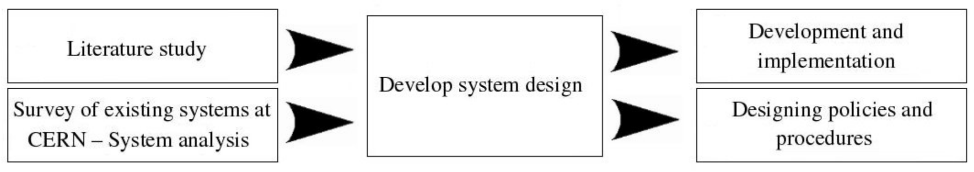

Literature study
Survey of existing systems at CERN – System analysis
Develop system design
Development and implementation
Designing policies and procedures

Figure 1: The process of the project

[1]http://ieeexplore.ieee.org
[2]http://portal.acm.org

In parallel with the next part of the development phase, the development of scripts, was also the design of a key management and administration policy and procedures. Different ideas had to be discussed with the relevant people at CERN, and different opinions had to be taken into consideration.

The resulting system design is presented in chapter 5, the results of the development phase are presented in chapter 6. The policies are presented in chapter 7, followed by the procedures in chapter 8.

3 Related work

Several solutions to similar problems have been presented in the past. After a presentation of different vulnerabilities of such backup systems, this chapter will present different cryptographic file systems, different file encryption programs (on-the-fly encryption), the writing of policies, ways to handle the key management, and disaster recovery.

3.1 Threats and vulnerabilities

One of the main abilities of a backup solution such as the one presented in this report is that the information may be stored over a long time period. This opens for several vulnerabilities, in addition to those relevant for general purpose storage systems. Storer et al. [8] present several vulnerabilities concerning secure storage, and many of them are applicable to this situation. It is vital to understand these threats to make a long term storage system work.

One example of such a threat is the key management when considering encrypted file storage. The encryption keys need to be stored for an indefinite time period. Compared to a single session key corresponding to one single transmission, this opens for several security threats. This is one of several problems in the area identified by Storer et al. as "long-term secrecy". Other examples are the problem with re-encryption and the fact that nobody knows the future development.

Another area of problems is that of authentication and user accounts. What happens when an employee leaves the organization, or for some other reason is no longer available? It is necessary to have proper mechanisms in place so that when e.g. an employee leaves his/her job, the user(s) taking over his/her responsibilities receives the necessary privileges and keys. In addition to this, it is important that the users remember where they stored the information, or else it is as good as lost.

The integrity of stored data is very important, and this is also crucial for long-term storage. It is therefore necessary with regular integrity checks to ensure the quality of the information stored in long-term storage systems. Storer et al. asserts that cryptographic hashes may not be sufficient for integrity checks in these situations, since an adversary might find collisions in the hash function and thereby alter the file without the integrity check noticing the change. This leads to another problem, which is the increased possibility of slow attacks on a long-term storage system. An attacker may use several years to complete an attack. Intrusion detection may be hard to accomplish, since the attack may be distributed over a long period of time.

Graf and Wolthusen [9] also present different threats to a storage system. They divide them into on-line and off-line threats, where on-line is defined as "occurring while a trusted OS is providing mediation to storage resources", and off-line is when there is "no trusted mediation of access to storage resources." Examples of on-line threats are impersonation, subversion (making privileged users perform the desired action) and privilege escalation (gain access to privileged resources accessible only to the OS). An example of an off-line attack is theft of removable media, e.g. a flash disk or even a laptop.

Riedel et al. [10] present a framework for evaluating the security of a storage system. Their framework consists of five components, spanning from the different users of the system to the granularity of the protection. The most important factors of their framework in this situation are:

- the users; file owners and those with read and write access,
- attacks,
- security primitives; authentication, authorization, key distribution and data protection.

It is also relevant to consider the user inconvenience – what level of inconvenience are the users willing to tolerate? This is important to see in connection with the desired level of security. Whitten and Tygar [11, 12] state that security software is only usable when certain criteria are met: the users should be aware of the necessary security tasks, they should be able to find out how to perform these tasks, no dangerous errors should be made, and the users should be comfortable enough with the system to use it again. Even though their work is mainly focused on the user interface of the system, the same principles will apply to a command based system.

To use the situation covered by this thesis as an example: if the scheme used to encrypt the information is too difficult for the users to understand, or in other ways create too much inconvenience, the users may avoid using the solution, and just use the onsite backup. As stated in [11], "if security is too difficult or annoying, users may give up on it altogether." In this situation, this will of course cause a massive loss of data in the event of a disaster.

3.2 State of the art

When storing data on a remote server, it is often desired to encrypt these data, either because the server is inherently untrusted, or because there is a chance that the server may be compromised [10]. As stated by Blaze [13], the use of remote file systems opens the possibility for an adversary to gain access simply by compromising a single component of a large system. This is the main reason of implementing encryption – it makes the data unreachable to those that do not have the correct key.

Here we will present some possible solutions to this problem, to show how similar situations have been solved in the past. At the end of the section there will also be a presentation of the tool implemented in the solution of this master's project.

CFS - Cryptographic File System

CFS was presented by Blaze [13] in 1993 and was "designed on the principle that the trusted components of a system should encrypt immediately before sending data to untrusted components." He states that the system is situated somewhere between the low-level (the cryptographic services are part of the system) and the user-level (the user directly controls the encryption/decryption) cryptography. The purpose of the system is to "protect exactly those aspects of file storage that are vulnerable to attack", and that the use of the system shall be convenient enough to be used on a regular basis. Some of the design goals of the system were:

- rational key management and natural key granularity;
  the user should not need to enter the key more than once. Also, it should be easy to protect related files using the same key.
- protection of file contents;
  in addition to the data in the files, the structural data related to the file's contents should also be protected.
- protection of sensitive meta-data;
  the (possibly sensitive) file names and other structural data should be protected as much as possible, and should not be readable without the correct key.
- protection of network connections;
  it should not be possible to read data by observing the network traffic.
- compatibility with underlying system services;
  e.g. administrators should be able to back up the data without knowing the key and without special tools.
- portability;
  encrypted files should be possible to read wherever and whenever the key is supplied, and the system should not be based upon special-purpose or otherwise unusual system features.

To encrypt a directory using CFS, the user issues a simple *attach* command. When writing files to this directory, these will then be automatically encrypted. Likewise, the file is decrypted when it is read. This is done using a "virtual" file system, where entries associating cryptographic keys with directories residing anywhere in the system name space are created. The virtual file system is typically mounted on /crypt, and to the user originally issuing the *attach*-command, all the files appear in cleartext here.

The keys can either be entered by the user directly through the keyboard, or be provided by hardware, such as smart cards. When using the keyboard, the key is generated from arbitrary-length passphrases. If the system utilizes smart cards, the keys are copied from the card to the computer after the user has entered his password to get access to the card itself.

The user may then create files and directories as usual, and when he is finished, he can simply detach the /crypt mount point. The underlying directory will of course still be there, but in encrypted form, and it must be attached at a later time to get the data in cleartext.

CFS uses DES to encrypt the data. Obviously, the development of the cryptography has, as any other part of the computing world, been substantial since 1993. Today, DES is not recommended to be used for encryption, because its relatively short key (56-bit) makes it vulnerable to brute-force attacks. This is especially not suitable for a long-term backup solution.

SFS – Secure File System [14]

The SFS "provides end-to-end encryption and key management support to users accessing file across the Internet on HTTP or FTP servers" [14]. The system was developed to be independent of operating system and applications. The access rights to a file are kept with the file as an attribute. The system supports decentralized access control, so that small groups of users can define who is allowed to read the data, without the help of a system administrator. With SFS, the information is protected both when it is stored and when it is transferred. As stated in [14]: “SFS encrypts from the information producer to the information consumer”.

SFS uses a Group Server to manage the keys used to encrypt files. In addition to this, each user has a smart card where his private keys are stored. These cards are used to digitally sign and encrypt all communication with the Group Server.

A header is stored with every encrypted file. This header holds an access control list related to the file, as well as the public key of the Group Server. When a file is accessed, the access list is forwarded (signed and encrypted) to the Group Server, which then determines whether the attempted access should be allowed or not. If the access is permitted, the server returns the file key to the user in encrypted form. Using his private key from the smart card, the user may now decrypt the file key, and is further able to decrypt the file. All encryption and decryption is executed internally in the smart cards.

To “provide fine-grain control of file accesses” [14], SFS uses separate keys for each file. Also, the directories are treated as files, so they have unique keys as well.

Farsite [15]

Farsite is “a serverless distributed file system that logically functions as a centralized file server but whose physical realization is dispersed among a network of untrusted desktop workstations” [15]. The goal while designing Farsite was to manage the unreliable machines of a network to provide a file-storage service that was logically centralized, secure and reliable. Cryptography and replication techniques are used to protect file data and directory metadata.

Since the Farsite system is developed to run on the workstations of large corporations or institutions like a university, it is not directly applicable to our situation. Furthermore, it is not suited to function as a backup service. However, it is nevertheless useful to investigate how the key distribution and management is solved.

To manage the trust issues in distributed file system, Farsite uses certificates. The certificates are signed using private keys generated for that purpose, and each certificate holds the public key of the relevant user. When the certificate is signed, the user name is associated with the public key in the certificate. Also, each machine has a public/private key pair and a certificate connecting the machine to the public key.

Farsite encrypts both the contents of the file and the metadata. A symmetric file key is generated when a client creates a new file. This key is used for the encryption, and further the key itself is encrypted using the public keys of all users authorized to read the file. Finally, the encrypted keys are stored with the file, so all users with a corresponding private key are able to decrypt the file key and further decrypt the file itself.

TCFS – Transparent Cryptographic File System [16]

One thing which is desirable at CERN is that the solution is transparent to the user, except for the process of providing the user credentials. CFS was the initial motivation when TCFS was designed. Functionality such as transparency and the possibility to choose which file in the directory should be encrypted, instead of encrypting all files in the directory, was added.

When an application requests a block of data, this is sent in encrypted form from the server. The client workstation decrypts the data before it is passed to the application. Similarly, if an application is to write something to the server, this is encrypted on the workstation before it is sent to the server, where it is stored. This way, there is no need for the server to be trusted, because it never handles the data in cleartext. Also, the data is always encrypted when sent over the network.

TCFS users generate their own keys, which then are stored in encrypted form in a database where the user's login password is the key. The users need to be registered as a user of the key database to benefit from this. To access his files, the user extracts his key from the database and presents it to TCFS. Finally, the user himself needs to run a command to make sure the key is erased from the kernel when he is finished.

There is also a solution utilizing Kerberos, where the users first authenticate themselves and receives a session key and a ticket. The ticket, and a request (e.g. to get a user key), is then sent to a TCFS key server (encrypted), where the message is decrypted and responded to.

Each file is encrypted with a randomly chosen file key, which again is encrypted with the user's master key and stored in the file's header. The files are encrypted in blocks – each block by a different block key. These block keys are created by hashing the file key concatenated with the block numbers. Each file block can also hold an authentication field, which is computed by hashing the block data concatenated with the block key.

This modular encryption scheme makes it impossible to break the encryption by checking if two encrypted files, or if two blocks in the same file, correspond to the same cleartext. The authentication fields also ensure that modification of the data on the server is detected.

OpenSSL [17]

Thompson [18] presents a solution for encrypted backups using scripts. His first approach is to use OpenSSL. OpenSSL is an open source toolkit that enables the user to implement the SSL (Secure Sockets Layer) or TLS (Transport Layer Security) protocols, as well as providing a strong general purpose cryptography library.

The cryptography library provides the users with a wide range of algorithms, and provides functionalities such as symmetric and asymmetric cryptography, cryptographic hash functions and pseudo-random number generators. Examples of available algorithms are blowfish, DES, IDEA, RC4, RSA, HMAC and SHA. OpenSSL also supports certificate handling, with x509 and x509v3 certificates.

Thompson's approach when using OpenSSL for backup encryption is suited for those already using scripts to carry out their backup services. By adding an extra OpenSSL command to the script, the data will be encrypted before it is backed up. Similarly, a command added to the script for restoring the backups will enable the decryption of the data.

GPG – the GNU Privacy Guard [19]

Thompson's second approach is to use GPG, or GnuPG. GPG is a free implementation of the OpenPGP standard presented in RFC2440[1]. It gives its users the ability to encrypt and sign data and communication. It is a command line tool, and supports algorithms

[1]RFC2440 – OpenPGP Message Format [20]

such as ElGamal, RSA, AES, 3DES, Blowfish, MD5 and SHA-1.

As with OpenSSL, Thompson includes the GPG encryption/decryption commands in the backup scripts. Even though GPG originally is a public-key (asymmetric) system, it is also possible to use symmetric algorithms such as AES to encrypt the data.

Neither the OpenSSL nor the GPG solution would be easy to implement in a system that requires incremental backup, since each part is encrypted before the backup is stored. To make the backup incremental, the existing backup file would then have to be restored, decrypted, the files synchronized and the new backup file encrypted.

Amanda

Amanda was developed in 1991, to solve the problems regarding backups from client workstations to backup tapes, especially the need to get the backup done between the end of one work day and the beginning of the next [21].

According to Garcia and Pragin [22], Amanda is “flexible, secure and scalable to dynamic computing environments.” It has been used in all kinds of environments, from a standalone machine to systems with hundreds of clients. The software gives the system administrator the ability to back up “multiple networked clients to a tape- or disk-based storage system” [23]. Also, the system is claimed to be easily and rapidly set up.

The system uses standard operating utilities such as dump and GNUtar, and this makes it possible to recover the data at any time with standard tools that are always available [24]. Encryption of the backups can be done with symmetric or asymmetric encryption algorithms, either on the client- or the server side. The system administrator may specify which encryption program to use, and a sample program is provided with the software. This sample program for symmetric encryption supports AES-128, AES-192 and AES-256, using SHA-256, SHA-385 or SHA-512 respectively [25].

Amanda was originally developed for Linux and Unix platforms, and is available for several of these. However, since October 2006, an installation package for Microsoft Windows is also available [26].

dm-crypt/cryptsetup [27]

dm-crypt is included in the Linux kernel, and is a tool to create “layers of virtual block devices” and encrypt these devices. Cryptsetup is a command line user interface used to create and manage such encrypted devices. Upon creation of an encrypted partition, the user can specify such things as encryption algorithm, hash algorithm and key length. The hash algorithm is used to create the encryption key from the passphrase provided by the user.

Cryptsetup and dm-crypt have some functionalities which are advantageous, such as the possibility to resize the devices and the support of keyfiles. However, the available documentation is very limited. In addition to this, it is necessary with root (administrator) privileges to create new devices. Based on this, cryptsetup is not considered as the overall best tool to use for this project.

BestCrypt [28] and TrueCrypt [29]

BestCrypt and TrueCrypt create virtual, encrypted disks and mount these as real, plain text disks [28, 29, 30]. The functionality of the two is very similar, so we will first give a general introduction of the concepts.

First, a volume, or container, is created. This may be an ordinary file, which can be copied, e-mailed, removed etc. just like any other file, or it may be a device, e.g. a USB flash disk, or a hard disk partition [31]. In these volumes, the data is stored in encrypted form. The user only has to provide the key once to access the encrypted contents within the volume. The functionality is similar to that of CFS, as the user mounts the volume on a desired mount point, and, after providing the appropriate user credentials, all the data in the volume is available as plain text to the user.

After mounting, the files can be accessed as any other files, and any modification, removal, synchronization, etc., of files can be done. The software provides so called "on-the-fly-encryption", where the data is automatically encrypted without any user intervention. When data is read, it is decrypted in memory before it is displayed, and when data is written, it is encrypted in memory before writing to disk. Data is never stored in plain text [28, 29].

Both TrueCrypt and BestCrypt are available for both Linux and Windows platforms, and create portable platform independent volumes.

BestCrypt is available on the Internet, and is called "open architecture". The source code is freely available, but there are restrictions on modification etc., and there is a license fee.

When a volume, or container, is created using BestCrypt, the password/passphrase for the container file is obtained from the user. BestCrypt then collects a random seed value from the Linux file /dev/random[2]. SHA-256 is then used to generate a key, using the seed from /dev/random as input. This key will be used to encrypt the data stored in the container. A key data block is created, where, among other things, the encryption key for the container file is stored. This key data block is then encrypted using the SHA-256 hash of the password from the user [33].

So, when a user requests to mount a volume, he or she is asked for the password, the key data block is decrypted using the hash value of the password; thereby the key to the data in the container is obtained. The key is then loaded to the Encryption Module, and the data is encrypted and decrypted on-the-fly [28].

Obviously, this can make BestCrypt vulnerable to brute force attacks, and it is highly necessary with long passphrases that are as random as possible.

TrueCrypt is open source software freely available to anyone. When a TrueCrypt volume is created, the software collects random data and generates random number sequences which are used to generate the master key and secondary master key used for the encryption. These keys are stored in encrypted form in the header of the volume. To decrypt these, a user must provide the correct password, and/or the correct keyfiles. The content of the keyfiles and the password are combined and hashed, and the resulting hash value is used to derive the header key through the function PBKDF2 (PKCS #5 v2, presented in [34]), using the hash algorithm specified by the user when creating the volume [29]. A simplified illustration of a TrueCrypt volume is shown in figure 2.

TrueCrypt has a random number generator that is used to generate the key to encrypt the volume, salt and random keyfiles. This generator "creates a pool of random values in RAM (memory)." This pool is 320 bytes long, generated using data from mouse move-

[2]"The character special files /dev/random and /dev/urandom (present since Linux 1.3.30) provide an interface to the kernel's random number generator. (...) The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. (...) /dev/random should be suitable for uses that need very high quality randomness such as one-time pad or key generation" [32].

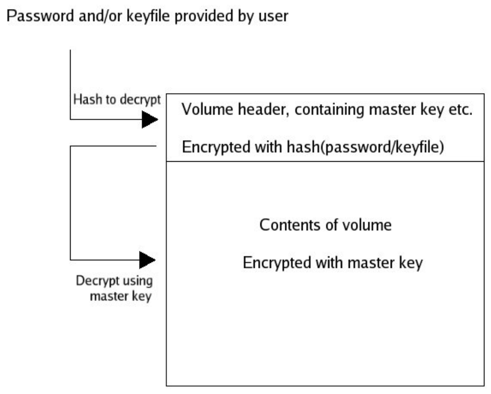

Password and/or keyfile provided by user
Hash to decrypt
Volume header, containing master key etc.
Encrypted with hash(password/keyfile)
Contents of volume
Encrypted with master key
Decrypt using master key

Figure 2: Simplified illustration of TrueCrypt volume

ments, keystrokes (on Linux only when the mouse is not connected to the computer) or values from /dev/random or any other specified source file.

TrueCrypt supports three encryption algorithms (as of March 2007) – AES-256, Serpent and Twofish, all with 256 bits key size and 128 bits block size, in LRW mode. Several combinations (cascade) of these three are also supported [29]. In the updated version released in March 2007, the support for the creation of new volumes encrypted by algorithms with blocks of 64 bits was removed, as the use of 128 bits blocks are more secure [35].

TrueCrypt also supports the use of keyfiles, which enables the user to store the key in a file instead of, or in addition to, entering a password. This is very useful in a system such as the one presented in this thesis, since it enables the use of scheduled, automatic backups.

We have chosen to use TrueCrypt to solve the initial task in this project. This is because the software meets the requirements set, and also because of the support of keyfiles, which is very suitable in this situation. Further reasons for this choice are given in section 4.4.

3.3 Writing policies

According to the Information Security Forum (ISF)'s "Standard of Good Practice for Information Security" [36], an organization should have a comprehensive and documented information security policy. This should be communicated to all relevant employees to "document top management's direction on and commitment to information security". Also, they state that the information security responsibilities of the staff should be described in staff agreements.

Bowden [37] describes a security policy as a "well written strategy on protecting and maintaining availability to your network and it's resources." He mentions several areas this policy should cover, including risk assessment, password policies, e-mail policies and disaster recovery.

According to Wills [38], "policies should be written to minimize the effort to maintaining them, yet be clear in the objective, boundaries and procedures to enforce them." It is also important to include possible exceptions to the rules, and clearly define these.

Policies that are system-specific, such as the one presented in this thesis, are often written with the particular product in mind. These policies may have to be revised more frequently than others, following the development in the technology.

It is important that the policies are written so that everybody has the possibility to understand its purpose, and they should be easily available to all the relevant staff [38]. The document should be meaningful, practical and inviting, address the users directly and be able to convince them of the necessity of secure handling of information resources. Different policy documents in one organization should follow the same guidelines, to create an overall communication style to the organization. Consequently, the information security policy and other policies relating to it should also be using the same style, to avoid the risk of being alienated [39].

According to Menezes et al. [40], a security policy should define the threats a specific system is intended to counter. Such a specific security policy is for example usually employed to provide the key management.

3.4 Key management

Menezes et. al [40] states that key management "is the set of techniques and procedures supporting establishment and maintenance of keying relationships between authorized parties". According to Schneier [41], it is a critical part of cryptography, and also the hardest.

The key management is important to meet relevant threats to the system, such as the possibility of the confidentiality and authenticity of keys being compromised, and possible unauthorized use of the keys. As previously mentioned, a security policy is a common way to present the key management. In addition to defining possible threats, such policies specify practices and procedures, responsibilities of different parties involved, and which different records to be kept [40]. The policy defines the organizational aspects of the key management, such as who is allowed to issue keys, to whom they may be issued, how the identity of those requesting a key is verified, how the stored keys are secured, which mechanisms are in place to control that the policy is being adhered to, etc [35].

In addition to the policy describing the organizational aspects of the system, other aspects must also be defined. How to handle the keys is a fundamental part of the key management [40], and this includes:

- key generation
- key distribution
- updating of keys
- revocation of keys
- destruction of keys
- storage of keys
- backup/recovery of keys

Many of these tasks may be handled and automated by the system itself, but the basic structure should nevertheless be defined. Although all of the tasks to some degree are important in all cryptographic systems, some are more relevant to this project than others:

Key generation is the process of creating the keys that are going to be used. This must be done randomly, or at least pseudo-randomly[3], to reduce the risk of an adversary being able to deduce the key. For symmetric cryptography, the output from the (pseudo) random number generator can be used as the key.

Key distribution is the task of placing the key where it is needed in the system. In some cases, this can be done personally – one person giving the key to another – in others, there is need for an encrypted communication channel to transfer the key.

Stored keys must be protected from access by unauthorized users. One way to do this is to generate the key from a password or passphrase that is sufficiently long but easy to remember. Another option is to use a trusted entity to hold and hand out the keys.

3.5 Disaster recovery

> The disaster recovery plan (...) describes the exact steps and procedures personnel in key departments, specifically the IT department, must follow in order to recover critical business systems in the event of a disaster that causes the loss of access to systems required for business operations. [43]

According to Fallara [44], disaster planning is smart for those who desire to protect valuable assets. The plan's purpose is to protect against anything that threatens "the function or existence of a business". This may be anything from earthquakes and tornadoes to a computer virus.

Before the disaster recovery plan is developed, it is important to identify the processes, resources and data that are critical for the organization. These must then be backed up to a remote location, so that they are available at all times. Then, the disaster recovery plan is formulated to define the strategy to recover those processes and data, and how to keep the production running as continuously as possible [43]. It is also important to establish the prioritization in case of a disaster, what should be recovered first?

For a system such as the one covered by this thesis, the disaster recovery plan should, among other things, cover:

- where are the keys stored
- who is in charge of restoring the keys
- how should the restoring of the backed up data be done

After the disaster recovery plan has been designed, it is important to run the necessary tests and to do the necessary training. This must be done to make sure that if there is ever the need for the disaster recovery plan, those responsible for different tasks are aware of their responsibility, and they know how to carry out the task. The testing is also important to be sure that the systems work properly.

Merkow and Breithaupt [43] assert that the CISSP, the Certified Information Systems Security Professional, identify five ways to test a disaster recovery plan:

- Walk-throughs: tracing the steps through the plan, looking for inaccuracies and things left out.
- Simulations: performing "dry runs" of an emergency, trying to make the response as close as possible to that of a real emergency.

[3]A pseudo random number generator takes as input a *seed* of binary numbers and produces a sequence of seemingly random bits as output. However, this output is not really random. [35, 42]

- Checklist: key personnel “check off” the tasks they are responsible for and report on the status. “This is typically a first step toward a more comprehensive test” [43].
- Parallel testing: “The backup processing occurs in parallel with production services that never stop.” [43]
- Full interruption: The systems are stopped as if a disaster has occurred, and the performance of the backup services is observed. If the test fails, the result may be as expensive as a true disaster would be.

4 System analysis

The system design proposed in this thesis is based on an existing offsite backup system available to the users at CERN. This chapter will present this system, and an analysis of the vulnerabilities of this system, which leads to the implementation of a secure backup system. The secure backup system design, which is presented in chapter 5, is based on several components already available in the existing computer infrastructure at CERN. These components will also be presented in this chapter. At last, there will be a brief presentation of TrueCrypt.

4.1 Offsite Backup Service

The Offsite Backup project at CERN was started in March 2006, to provide a service where a current copy of vital data could be stored. This is necessary as part of a disaster recovery plan, in the event that both the relevant production servers and the regular backup servers are destroyed. The service is not intended as any kind of archival backup service, and only holds a live copy of the most important data [45].

The offsite backup servers are located on the CERN site in Prevessin. This is about three kilometers from the Meyrin site, where the computer center with the different servers and the regular (archival) backup servers are located. The server is Intel based, with two Intel Xeon 2,66 GHz CPUs, 2GB RAM and disk controllers and disks providing ~3TB of disk space. The operating system is SLC4 (Scientific Linux CERN 4), which is a Linux distribution based on Red Hat Enterprise Linux [46].

The initial requirements to the system were:

- The system must offer the possibility of scheduled backups, but it must also be possible to accept backup data from client machines at any time.
- The system must be portable.
- The system must be easy to operate. The user must not need to enter a password to perform the backup.
- The data must be transferred securely.
- The data stored by one client must not be visible by any other client.
- The client must be able to restore the data at any time.
- The solution must be inexpensive, and must use as much CERN standard material as possible, and software used must be Public Domain.

A service account is created for each client associated with the offsite backup server. In this context, a client is not a single user (person), but a group or service that needs a current offsite backup. Consequently, the number of clients is relatively low. Using the service accounts, the clients may then transfer data to the backup server using SSH, e.g. by the use of scp or rsync over SSH (Linux). It is the clients' responsibility to build the set of files which composes the backup, to organize the data, and to regularly restore the backed up data to validate it. There is no global monitoring of the validity of the backups.

After the initial introduction of the system, the clients expressed a desire to have the backup stored on the server encrypted. This is necessary to secure the data against:

- physical attacks from the outside, e.g. if an intruder is able to steal the backup server
- attacks where an intruder gets physical access to the backup server inside the computer center, or is able to remotely access the server
- attacks from insiders

4.2 AFS – Andrew File System

When a new service account is created, this also includes the creation of an AFS account corresponding to the offsite backup client. AFS is a distributed computing environment. The development was first started at Carnegie Mellon University in 1983. It is now developed by the IBM Pittsburgh Labs, and was open sourced in 2000 [47, 48, 49]. The AFS client software is available for several UNIX platforms, including Linux, SUN and IBM, as well as for MS Windows and Mac OS X. “This makes AFS the ideal file system for data sharing between platforms across local and wide area networks” [50].

The purpose of a distributed file system is to “make access to distributed files indistinguishable from access to the local disk” [51]. In an AFS environment, data can be replicated to several locations. Read-only clones can be stored on different servers, and if one server goes down, the clients can transparently change to read the files from another server. This is also possible because the AFS system uses a globally unique name space – the client does not need to know on which file server the file is located, only the file’s pathname. The client then contacts a data location database to find the data. In this way, the system also allows administrators to move data from one server to another without the user noticing this.

AFS client machines store data copied from the server(s) in a local cache. This reduces network traffic, since the data does not have to be retransmitted over the network frequently. The data is requested from the servers by a cache manager process. To maintain consistency, “the file server keeps track of which clients have cached copies and tells the client to invalidate its copy should data change” [51]. This is done when a file is closed.

AFS also supports traffic encryption between a single client and the AFS server. This is enabled by a simple command, and do not require the client machine to be restarted when initialized. The encryption is done using an algorithm called FCrypt, and a description of this can be found in [52]. Basically, it is a DES based block cipher, using a 64-bit key and 64-bit block size. According to Marcus Watts from Michigan University, an AFS development contributor, the mode of operation is CBC, and the Kerberos session keys are used directly, without any further key generations. He asserts that the algorithm was probably obsolete already when it was first implemented in AFS, but its advantage was speed.

FCrypt is a weak encryption algorithm, and this is also discussed in chapter 9. However, Russ Allbery from Stanford University, who also contributes to the development of AFS, ensures that the area of traffic encryption is the top priority of the AFS development group, and that substantial progress has been made the past year. The correspondence with the OpenAFS mailing list can be found in appendix C.

For authentication and security, AFS uses Kerberos. This will be covered next.

4.3 Kerberos

Kerberos is a network authentication protocol, "designed to provide authentication for client/server applications" [43]. It was invented in the late 1980's at the Massachusetts Institute of Technology (MIT) in Boston, and have since then become a widely used key management system [35].

The main objectives of Kerberos are (from [42]):

- *Security:* No attacker should be able to impersonate someone else or eavesdrop on information.
- *Reliability:* The Kerberos service must always be available, since all use of a service requires authentication.
- *Transparency:* The user should need to enter his/her password only once in the beginning of a session, the rest of the authentication process should be transparent.
- *Scalability:* "Kerberos must have the ability to support a large number of users, workstations, services and servers."

Other requirements to the design were that (from [53]):

- The authentication must be two-way: The server can be certain of who the client is, and the client can, if it is required, be certain that it is the correct server.
- No cleartext passwords should be transmitted or stored on servers.
- On the client side, "cleartext passwords should be handled for the shortest possible time and then destroyed."
- Authentication has a limited lifetime

Kerberos is described in a number of books, such as [41, 42, 54, 55]. A brief description will be given here, more information is available in the referred sources.

Kerberos has a client/server architecture, where a client can be both users and software. The structure of the protocol is shown in figure 3. In this situation, user A wants to securely communicate with the server S. First, A authenticates himself to the system, obtaining a Ticket Granting Ticket (TGT). This is done by sending a request to the Kerberos authentication server, holding the client's identity and the identity of the corresponding Ticket Granting Server (TGS). After looking up the client in its database, the authentication server generates a session key to be used between the user and the TGS. This is called the TGT, and is encrypted with the user's secret key. The authentication server also creates a TGT for A to use when authenticating himself to the TGS, and encrypts this with the secret key of the TGS. The TGTs are returned to A, who is able to decrypt the session key using his password. The TGT and session key are valid for a given time period, e.g. 25 hours.

Now, A can get tickets for the services he wants to make use of, by sending requests to the TGS. Using the TGT from the authentication server (encrypted with the TGS's secret key), the TGS is able to authenticate the client. When the request is validated, the TGS returns a ticket for A to present to the server S, as well as a session key to be used between A and S, encrypted with the session key shared by A and the TGS.

A can now authenticate himself to S, by sending a similar message, holding authentication information, the ticket received from the TGS and the session key encrypted with

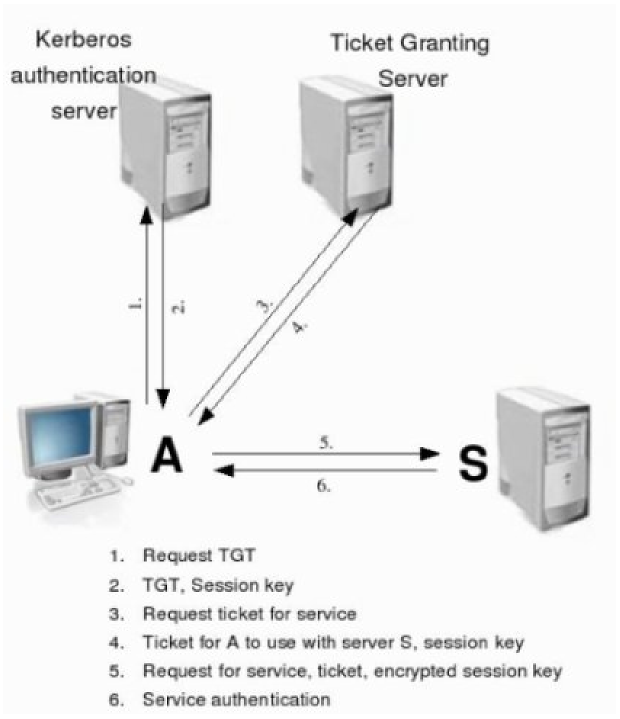

Kerberos authentication server
Ticket Granting Server
1.
2.
3.
4.
A
5.
6.
S
1. Request TGT
2. TGT, Session key
3. Request ticket for service
4. Ticket for A to use with server S, session key
5. Request for service, ticket, encrypted session key
6. Service authentication

Figure 3: Structure of Kerberos

S's secret key (also received from TGS). By validating this, the server knows that the user is who he says he is.

If mutual authentication is needed, the server then returns a message holding a time-stamp encrypted with the session key, proving that it was able to decrypt the previous message.

4.4 TrueCrypt

A brief description of TrueCrypt and its functionalities is given in section 3.2. As stated there, TrueCrypt is an open source software tool used to encrypt volumes of data. Virtual, encrypted disks are created, and these can be mounted as normal file systems, to be accessible in the same way as plain text disks. Here we give a short explanation of why TrueCrypt was chosen as the tool for this project, and how it is used. The TrueCrypt source code is available from [56].

As mentioned in section 3.2, dm-crypt/cryptsetup is a good alternative for this project. However, the lack of documentation makes it less attractive. That leaves BestCrypt as the main alternative. BestCrypt is a product similar to TrueCrypt, which is also available on the web. The choice was made in March 2007, and the main reasons why TrueCrypt was chosen were:

- TrueCrypt supported LRW mode of operation, while BestCrypt did not.
- TrueCrypt did not allow encryption keys shorter than 256 bits. This decreases the risk of a user creating a volume with a less secure algorithm without being aware of it.
- TrueCrypt supports the use of keyfiles.

- TrueCrypt is open source and free.
- The documentation of the Linux version of TrueCrypt is better than the documentation of the Linux version of BestCrypt. TrueCrypt also has a considerable user forum where it is possible to search for answers to possible problems.

The version of TrueCrypt used in this project is v4.3, released March 19, 2007, which is installed on the offsite backup server. A partition on the server is dedicated to the offsite backup, on which the TrueCrypt volume is created and formatted to ext3. The volume creation and formatting takes a substantial amount of time. The size of the partition used at CERN is 3725GB, and the creation of the volume, with formatting, took approximately 55 hours. There is, however, the possibility of a "quick" volume creation, but this is not recommended since it does not fill the volume with random data.

4.5 Vulnerabilities and assumptions

The main security issue for CERN, concerning the offsite backup server, is if an adversary gets physical access to the machine, e.g. by stealing it. In such a case, the information will be protected if it is encrypted.

There is always the risk of insiders attacking a system. In the system design presented in section 5.1 this is taken into consideration, by keeping the TrueCrypt volumes unmounted when they are not used. However, at CERN, it would probably be easier, and more advantageous, for a malicious insider to get access to the servers where the data is originally stored, in plain text, and perform an attack there (e.g. stealing or manipulating data). Therefore, it is at this point regarded as "safe enough" to have the volumes mounted all the time. Further information about this is given in section 5.2.

It is assumed that a user logging into a remote backup server via SSH from a machine in the CERN network, holding a valid Kerberos ticket that allows him to do so, is authorized to do this.

It is further assumed that the computers situated in the computer center on the Meyrin site are secure.

When a client requests the opportunity to encrypt the data stored on the backup server, it is assumed that all the data shall be encrypted. It is therefore presumed that there is no need for an opportunity to choose which files to store in encrypted form and which to store in plain text. When considering the design presented in section 5.1, it is also assumed that no one without the permission to decrypt the data needs access to any of it.

The users of the remote disaster recovery backup service are themselves responsible for validating the backup: checking that the backups are indeed carried through, that the correct data is backed up, etc. Hence there will be no integrity check or monitoring included in the system design.

5 System design

In this chapter, the system design for the secure offsite backup is presented. Section 5.1 presents the complete system design, which provides security based on the different users of the service, and the requirements presented in 4.1. Section 5.2 presents a simplified version of this system design, which is the one being put into production at CERN. This is more based on management by administrators and simplified use, and concentrates on the security of the backup if the backup server is stolen.

5.1 Secure Offsite Backup

The system for secure offsite backup makes use of several services that are already available at CERN (presented in chapter 4). Its focus is on securing each backup user's data, both from other users of the service and from outsiders. Another important focus is user inconvenience, to have as little interaction, and to make the solution as transparent, as possible.

The data, before backup, is stored on computers inside the CERN Meyrin site. As stated in section 4.5, these machines are regarded as secure. The servers providing AFS are also located on the Meyrin site, and these are also regarded as secure.

For this thesis, we disregard the aspect of space-limitations and possible maximum volume sizes. Initial tests show that TrueCrypt volumes (file containers) may be up to 2TB, and that TrueCrypt partitions seem to have no size limit. We therefore see this as irrelevant in the context of this thesis.

Figure 4 shows the components of the system, as well as the basic information flow when a user mounts a volume.

TrueCrypt is chosen as the tool to encrypt the data on the backup server. The reasons for this are given in sections 3.2 and 4.4. It was chosen to install the software on the backup server, leaving a limited number of machines to configure and maintain, at the time of writing only one. An alternative is to install the software on each client, and then mount the volumes locally on the client machines, but this requires the installation of additional software, and the configuration of, possibly, a vast amount of machines. The main advantage of installing TrueCrypt on every client machine is that the data are then sent in encrypted form and decrypted/encrypted in the memory of the client computer. Since it is necessary for the users to log in to the backup server using SSH anyway, this advantage is not seen to be outweighing the disadvantage of the increased maintenance.

The system uses the TrueCrypt keyfile functionality. This decreases the necessary interaction between the backup server and the user to a minimum, and it is also an extra security against such threats as keyloggers, since no password is entered.

The keyfile itself is stored in the service account's AFS account. AFS uses Kerberos authentication to mutually authenticate both the client and the server, by the use of tickets generated when the user log in [51]. Therefore, the keyfile will be inaccessible to anyone that is not logged in as the correct user.

To further secure the keyfile when transferred between the AFS server and the backup server, the traffic is encrypted. Traffic encryption in AFS is presented in section 4.2. The

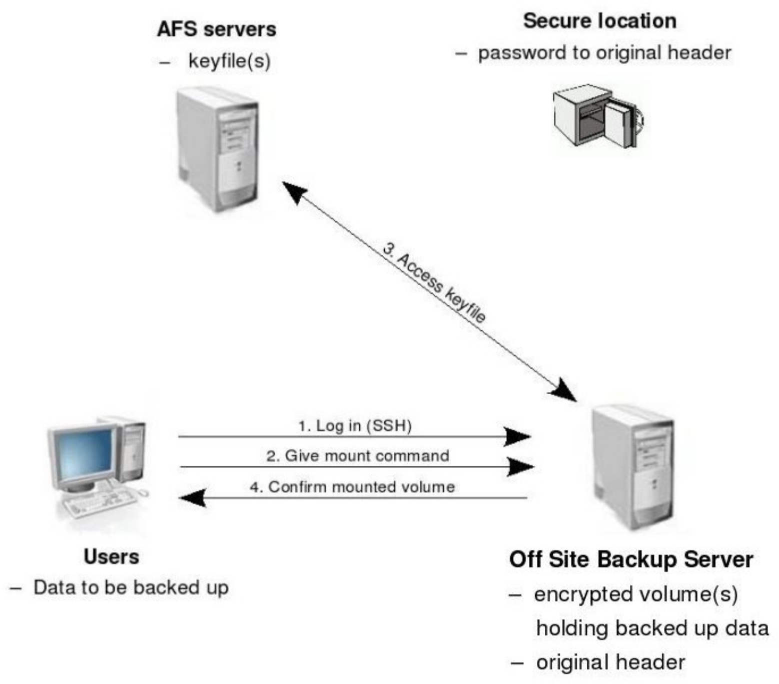


Figure 4: The system design; mounting a volume

encryption is turned on when the backup server is booted, and remains on at all times. Because the amount of traffic between AFS servers and the backup server will be very limited, decreased performance because of this is not an issue. Constant traffic encryption eliminates any possible uncertainty about whether or not the traffic is encrypted immediately after the encryption is enabled. As mentioned above, the encryption algorithm used is rather weak, and this is discussed further in chapter 9.

When a user wants to make an (interactive) backup, he/she uses either scp or rsync for this. Instead of using the commands directly, a script is called. Files/folders to be backed up are provided, either as parameters to the script or interactively when the script is called. Using SSH, the volume is then mounted, the requested backup command is run, and the volume is unmounted. To minimize the user inconvenience, the SSH connection is using cryptographic keys, so there is no need for the user to supply a password. Consequently, the session will, to the user, appear as when simply running scp/rsync, except for the actual command.

An automatic backup will be done in much the same way. The user, or an administrator, creates a backup script that will run at a given time or interval. This will contain commands to mount, copy/synchronize, and unmount the volume. Figure 5 shows the process when taking a backup.

For disaster recovery, the information needed to mount the volumes must also be stored somewhere away from the Meyrin site. There are four possible solutions:

1. To store the keyfile in a safe in a secure location.
2. To encrypt the keyfile with public/private key encryption, storing the public key on

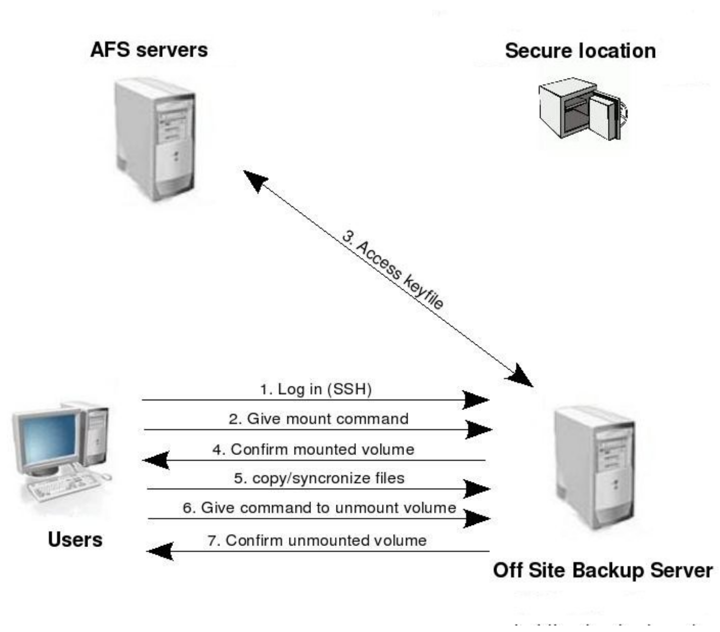


Figure 5: Process when taking backup

the backup server, and the private key in a safe in a secure location.

3. To encrypt the keyfile with symmetric encryption, using e.g. OpenSSL with a strong password, and store this password in a secure location.
4. Create a volume with a strong password. Backup the volume header and store the password in a secure location. Then change the volume so a keyfile is used instead of a password. In the event of a disaster recovery where the current keyfile is lost, the original header can be restored, and the volume can be mounted with the password retrieved from the secure location.

When focusing on decreasing user inconvenience, option 2 and 4 are those best suited, since these do not require any additional user interaction when changing keyfiles besides some simple key strokes. Option 1 makes it necessary to physically transport the keyfile to the secure location when it is changed, and option 3 makes it necessary to enter a password or passphrase.

For simplistic reasons, we have chosen to use the forth option, since that only requires creating a strong password/passphrase, and printing this, instead of integrating an asymmetric key encryption scheme. This also simplifies a possible disaster recovery, since it is only necessary to restore the original header and then use the stored password to change to a new keyfile. In this way, it is only necessary to use functions already available through TrueCrypt, instead of creating separate programs.

To secure against data corruption on the remote backup server, the backed up header file is also stored in the service account's ~/private folder in AFS. If both the header of the volume and the header file backed up on the backup server is corrupted, the copy stored in AFS can be used to restore the header in an attempt to rescue some or all of the data stored in the volume. However, the users of the offsite backup service are made

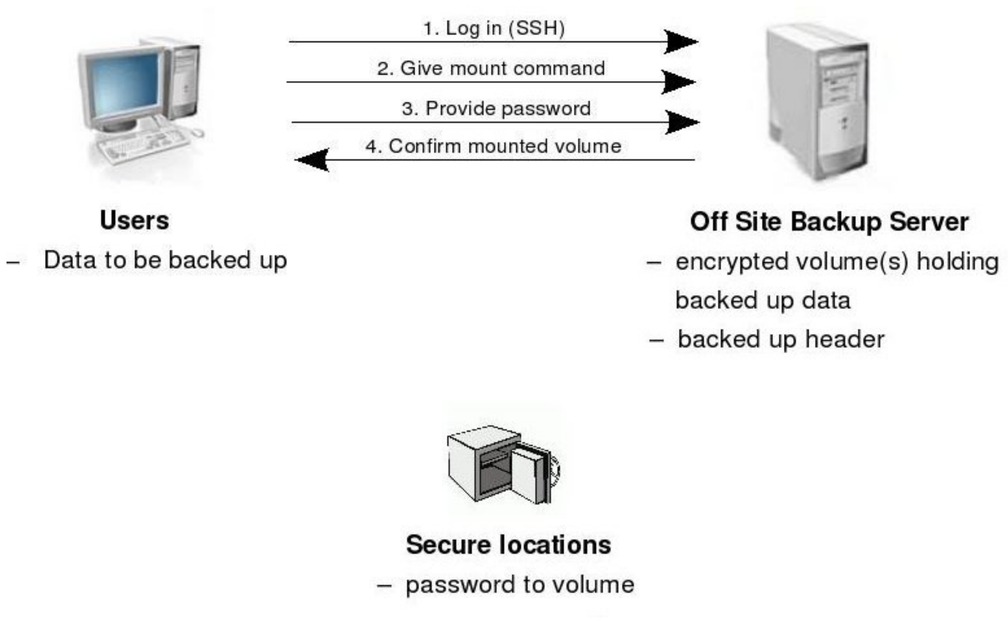


Figure 6: The version put in production at CERN; mounting a volume

aware of the fact that the data stored here are not further backed up and may be lost if the disk crashes.

To use header file and password stored in a secure location also secures against the loss of the keyfiles stored in AFS. If this occurs, the password can be retrieved, the header restored and the volume mounted using the password.

5.2 Simplified system design

After several discussions with the responsible section at CERN, it was decided that a simplified version of the system design was more suitable for the current situation in the organization. This section describes the system put in production at CERN. TrueCrypt is still installed on the remote backup server, as in the solution described above, and is used to encrypt one large partition. It was decided that there should only be one large volume, and that each user should have their dedicated area on this volume, where access was controlled using the normal Linux file permissions. For this to be possible, it was also necessary to format the volume to ext3, since FAT do not support these file permission. This means that administrator privileges are needed to create a new volume, but since this will be done by offsite backup administrators, this is not a problem. Secondly, it was decided that it was sufficiently secure to keep the volume mounted at all times. The reason for this decision is that all the information stored on the backup server is also stored in cleartext elsewhere in the CERN network, both the original files and in other backup locations. There is therefore no need to protect this backup any better. It is assumed that those being able to get access to the server this way are also able to get access to other cleartext versions of the data.

Following this, it is only necessary to manually mount the volume when the backup server is rebooted. This could then be done by a person with administrator privileges in the server, and it is not necessary for any of the ordinary users to be affected at all. It also decreases the demand for less user inconvenience, since it is not likely to happen very often, and since administrators and IT personnel often will be more willing to such things as offering an extra password. Therefore, it was chosen to use only a password for the TrueCrypt volume, instead of a keyfile.

Figure 6 shows the components of the system put in production at CERN, as well as the basic information flow when a user (*obadmin*) mounts a volume.

The choice of using a password eliminates the use of AFS, and keeping the volume mounted at all times makes the system totally transparent to the users. The only advisable change in their procedures is a simple check in the beginning of the backup scripts of whether or not the volume is mounted correctly. (If it is not, the backup should be aborted, the user alerted and a message sent to an offsite backup administrator.)

This solution gives the system administrators some more responsibility for the secure operation of the system. When creating a volume, it is necessary to print or write down several copies of the password. One must be kept safe and available for those system administrators that might need them (e.g. locked in a drawer in somebody's office), and the others must be kept in secure locations away from the Meyrin site. It is desirable to employ the same solution here as in the system design presented above, by using a safe to store the password to use for disaster recovery. However, CERN is composed of a number of different departments and groups, and it would be difficult to introduce policies and procedures that would ensure that no one without the right privileges would get access to the password stored in the safe. Therefore, it was decided that the offsite backup administrators would be responsible for securing the different copies of the password. This introduces some shortcomings both when concerning disaster recovery and general security, and this will be discussed in chapter 9.

For disaster recovery, the same approach is taken as in the complete solution. The volume header is backed up, and in the event that the copy of the password stored locally is destroyed, one of the other copies must be retrieved. If the password has been changed since the creation of the volume, the volume header must be restored before the volume can be mounted. If not, one can simply use the obtained password to remount the volume. In this way, the password of the volume may be changed without the need of printing and redistributing the new password to the holders.

As with the system design presented above, the use of password and header for disaster recovery ensures the ability to recover the data stored on the encrypted volume if the current password is lost.

6 Development and implementation

This chapter describes the development phase in more detail. First, a description of the process of the system design and installation of the tools to be used is provided. Then the results of the script development process are described. The design of policies is presented in chapter 7, and procedures in chapter 8.

As mentioned in chapter 2, the process of the project followed an iterative-incremental approach, which means that one can go back to an earlier phase of the project to change something or correct an error, and that the requirements to the system, and how to meet these requirements, are continuously revised with those who are going to use the system.

When the work on this project was commenced, the requirements to the functionality of the system were already established. However, it was important with a dialog with the users when establishing the system design and the key management scheme. This was done in several iterations. First, some proposals of which tool to use for the encryption were prepared, with a recommendation, and this was presented to the users (mainly offsite backup administrators). Feedback was given, and a decision was made based on this. The resulting proposal was also presented to personnel responsible for the computer security at CERN. At this point, it was decided that there was only need for one large volume, and that the standard Linux file permissions were sufficient to control the access to the data.

The next step was the key management scheme; first, a proposal was made and presented to the offsite backup administrators, then to the users of the system, and at last to the computer security personnel. The feedback from the different sessions resulted in the decision that since only offsite backup administrators would mount the volume, it was sufficient to use a password for this, instead of a keyfile. This again led to the alteration of the system to the simplified version presented in section 5.2.

The first part in the development phase was to install TrueCrypt on the backup server. This did not involve any major problems. Since there was no version available that would install on SLC4, the source code had to be compiled, and a kernel header file had to be added to the kernel source code folder. This was an issue reported by other users too, who were trying to install on Linux distributions based on Red Hat Enterprise (e.g. Fedora). After this, the software was compiled and installed without problems.

Some time was used to study the tool and test the speed of transmission and that there was no problems writing large amounts of data to the volume. Some troubleshooting was necessary when creating and formatting large volumes, to eliminate an issue of the computer freezing and making it necessary to reboot. Because of the good on-line documentation and the active user forum provided by TrueCrypt[1], these problems were solved fairly quickly.

When the software was installed and the necessary testing was done, the development of a command line-based user interface was commenced. Scripts are used to decrease the need for user interaction, and to decrease the users' necessary knowledge of the

[1]http://forums.truecrypt.org

system and the command syntax. This was the main part of the development phase of this project. Here too, there are some differences between the complete system and the system put in production at CERN. The theory behind the scripts used in the complete system is presented in section 6.1, while the differences and the scripts used at CERN are presented in section 6.2. The scripts are provided in appendix A.

6.1 Scripts for the secure offsite backup service

There are two main functions available in this system, and that is to create new volumes, and to initiate backups to already existing volumes. Both functions also need to be able to mount and dismount a volume, and to control that these actions are successful. In addition to this, it is necessary to have scripts to change the keyfile and to restore the volume header. In this section, the script to create a new volume is presented first, followed by the script used to check whether a volume is mounted, the scripts to mount and unmount a volume and the script to create the random password. Then, the script for taking a backup using rsync is presented, and finally, the scripts for changing keyfile and restoring the header. It is also necessary with a script on the backup server to read from the configuration file when taking a backup. This will not be presented here.

createvol.sh

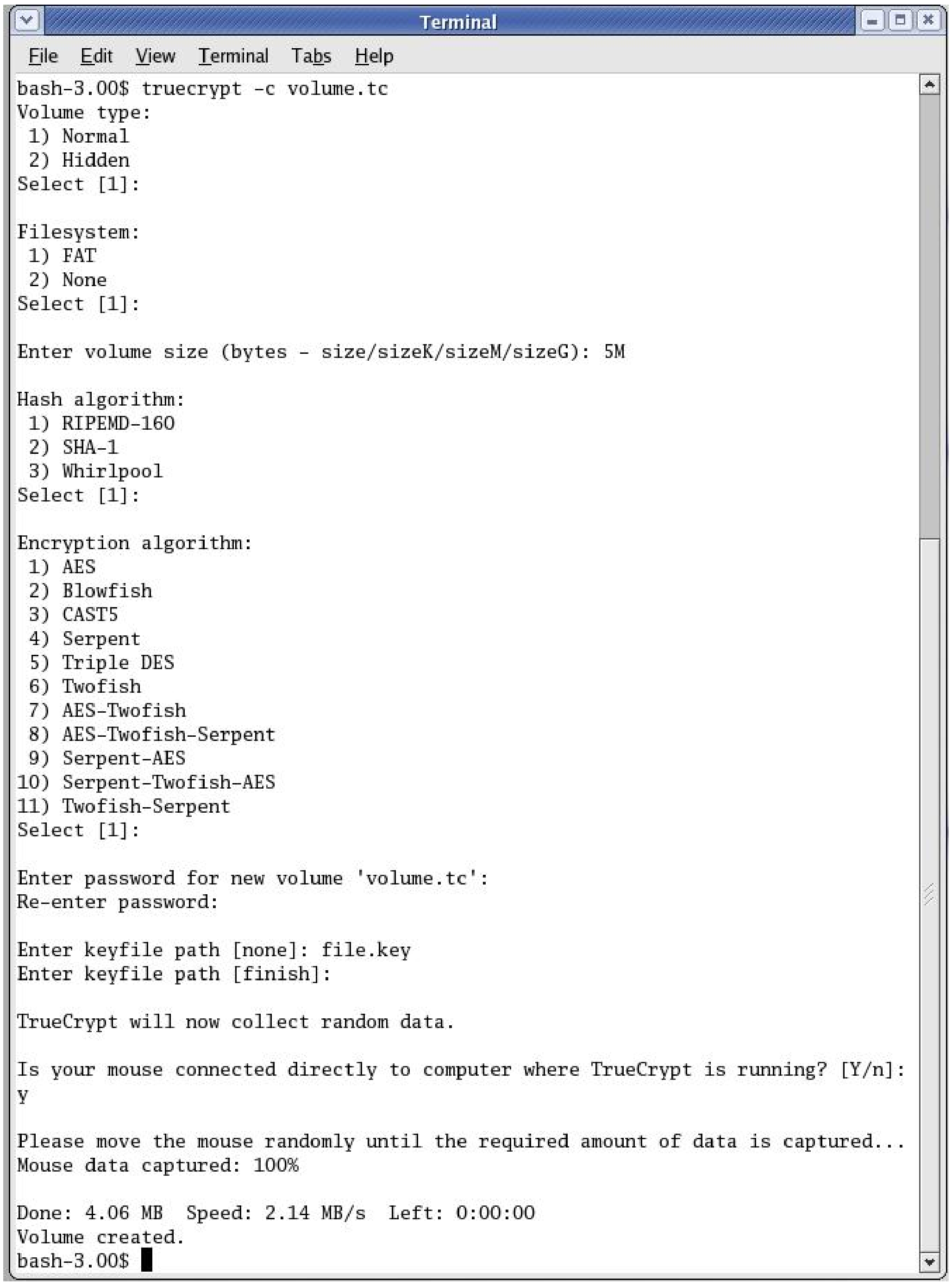
The creation of a volume requires much information that in many cases should be decided by the system administrators. Examples of this kind of information is the name (and path) of the volume and the path of the keyfile. Other information needed is volume type, size of volume (if it is not a partition), file system and which hash- and encryption algorithms to use. It is also necessary to collect random information, and this is normally done by mouse movement or keystrokes. Figure 7 shows the normal process of creating a TrueCrypt volume.

This is a lot of information for the users to learn and to remember. Also, a password must be generated, and a configuration file holding volume path and mount point must be created. In addition to this, the header must be backed up, a keyfile must be created and the volume must be changed so that it is mounted with the keyfile instead of the password. Because the password and header is used for the disaster recovery, it is necessary with some user interaction in this script. The TrueCrypt command for changing password/keyfiles also requires input from the user.

This script has five main tasks:

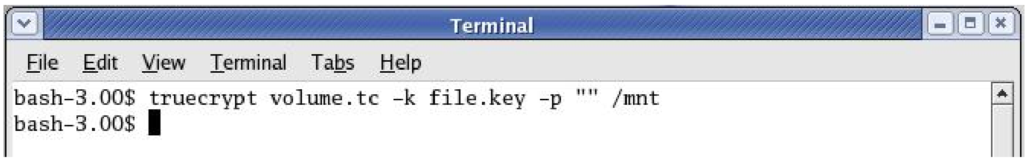
- Create volume using password from *random.sh*
- Back up volume header
- Create configuration file holding information about volume name and mount point
- Create keyfile
- Change volume to use keyfile instead of password

In this script, it is already decided which algorithms should be used for the hash and the encryption. In other environments it may of course be desirable for the users to make this decision themselves, and the options may then be removed from the command. TrueCrypt will request the options that are not stated in the command interactively.



```
bash-3.00$ truecrypt -c volume.tc
Volume type:
 1) Normal
 2) Hidden
Select [1]:

Filesystem:
 1) FAT
 2) None
Select [1]:

Enter volume size (bytes - size/sizeK/sizeM/sizeG): 5M

Hash algorithm:
 1) RIPEMD-160
 2) SHA-1
 3) Whirlpool
Select [1]:

Encryption algorithm:
 1) AES
 2) Blowfish
 3) CAST5
 4) Serpent
 5) Triple DES
 6) Twofish
 7) AES-Twofish
 8) AES-Twofish-Serpent
 9) Serpent-AES
10) Serpent-Twofish-AES
11) Twofish-Serpent
Select [1]:

Enter password for new volume 'volume.tc':
Re-enter password:

Enter keyfile path [none]: file.key
Enter keyfile path [finish]:

TrueCrypt will now collect random data.

Is your mouse connected directly to computer where TrueCrypt is running? [Y/n]:
y

Please move the mouse randomly until the required amount of data is captured...
Mouse data captured: 100%

Done: 4.06 MB  Speed: 2.14 MB/s  Left: 0:00:00
Volume created.
bash-3.00$ █
```

Figure 7: Creating a TrueCrypt volume.



```
bash-3.00$ truecrypt volume.tc -k file.key -p "" /mnt
bash-3.00$ 
```

Figure 8: Mounting a TrueCrypt volume.

TrueCrypt also requests random data to use as seed in the key generation and when generating the keyfile. Normally, this is done with mouse movement, or, when there is no mouse connected to the computer, keystrokes. In this situation, since the volume is created on a remote machine, this input would be keystrokes. There are two possible problems related to this: The user may make the wrong choice and answer “yes” when asked if there is a mouse connected to the computer, and thereby need to restart the whole process, or the same letter may be typed 320 times as input from the keyboard. To avoid this, the commands are run using /dev/urandom (for keyfile creation) or /dev/random (for volume creation) as random seed. This also reduces the need of user interaction.

This script always formats the volume to FAT, because that does not require administrator privileges (root) to the computer where the volume is created. However, it may be desirable to format the volume to e.g. ext3. Even though this requires root privileges, it is done in the system in production at CERN, and the script used for this is presented in section 6.2.

chech.sh

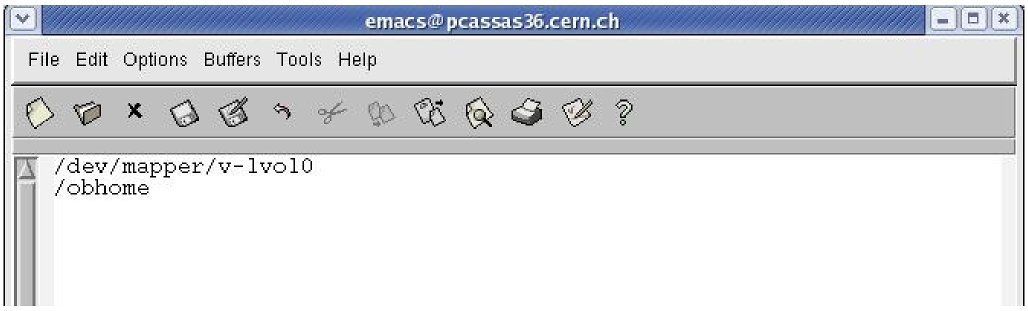
This script performs a simple check of whether or not a specific volume is mounted, based on the standard TrueCrypt listing command (*truecrypt -l*). Listing the mounted volumes, the script checks if the relevant volume is represented in the list. The script is called from other scripts, with the username as parameter, and the result is read into a variable. *chech.sh* returns 1 if the volume is mounted and 0 if it is not.

mount.sh

Figure 8 shows the command to mount a TrueCrypt volume using a keyfile and an empty password, without any further interaction. This command requires that the user knows the volume name, the name and path of the keyfile, and the desired mount point, in addition to the syntax of the command. In an environment such as CERN, it is a possibility that different people will be making backups to the same service account on the offsite backup server. It is therefore a benefit if all prospective individuals do not have to know all these details.

To solve this, there is a configuration file stored in the AFS ~/private folder of the service account, holding the volume name and the mount point. The structure of this file is shown in figure 9. The keyfile, which is also stored in this folder, has a name that corresponds to the volume name. The script then reads the configuration file, and fills in the relevant parameters in the command. The user can in this way mount the volume simply by running this script. The keyfile and configuration file are created when the volume is created, see the paragraph about *create.sh*.

Before mounting the volume, the script checks that it is not already mounted. It also controls that the mounting was successful. In addition to this, the utilized space on the



```
/dev/mapper/v-lvol0
/obhome
```

Figure 9: The structure of the configuration file.

volume is checked, and if this is over a defined threshold, the user and the offsite backup administrators are notified of this.

unmount.sh

To unmount one specific volume, the user either needs to know the mount point, the device number or the name of the mounted volume. It is also important to know the correct syntax, so that no other volumes are unmounted. By using a script to enable the users to unmount, one also avoids users using the standard Linux “umount” command, which would leave the volume mapped to the device, and thereby still accessible, unencrypted, without the keyfile.

In the same way as *mount.sh*, *unmount.sh* collects the necessary information from the configuration file, and runs the necessary command to unmount the volume. In addition to this, it calls *check.sh* to control that the volume was unmounted properly.

random.sh

This script simply generates 20 random characters, from a field defined in the script, and prints these. If *random.sh* is called from another script, the resulting string is returned to a variable.

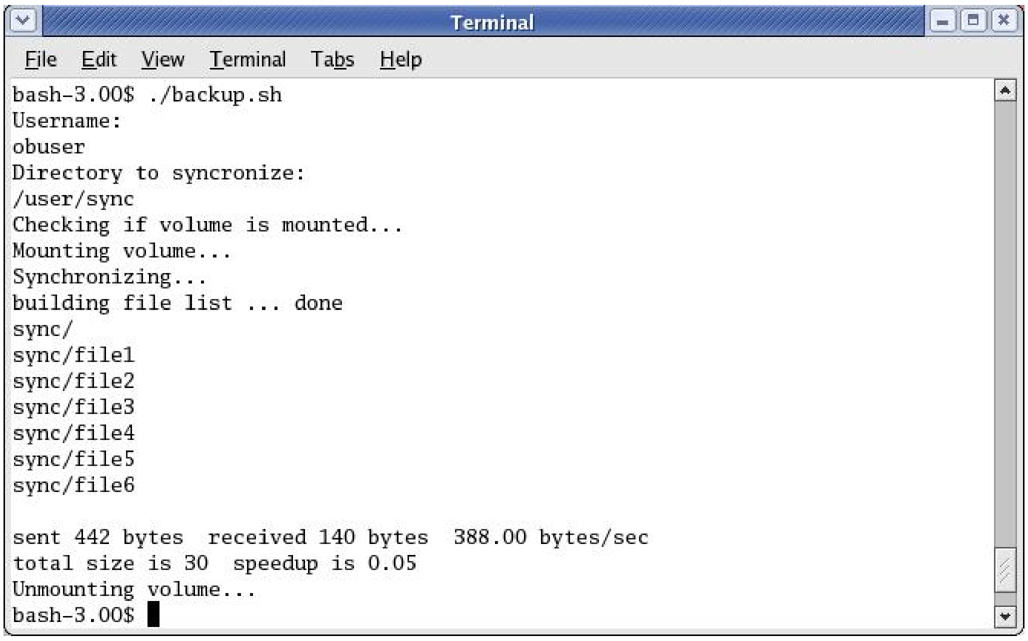
backup.sh

This script takes as input from the user the files/directories that should be backed up (synchronized) using rsync. SSH is used to log into the backup server and mount the volume. Then, rsync is run, synchronizing the requested files. When finished, the volume is unmounted. The scripts *mount.sh* and *unmount.sh* are used in the process, and these again use *check.sh* to see that their tasks were performed properly.

Figure 10 and 11 show the process when a user takes a backup using *backup.sh*. Figure 10 shows a backup where the user does not send the parameters when calling the script, and is prompted for the necessary information, and figure 11 shows the same backup, but here the necessary information is provided as parameters.

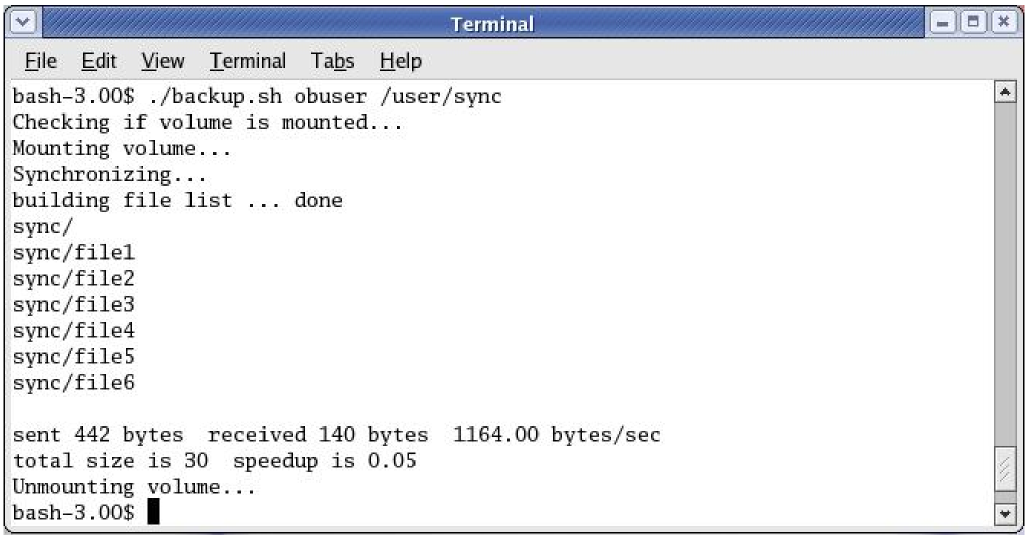
A similar script could be developed to do the same with scp – in that case the files would be copied instead of synchronized. This will, however, not be presented in this thesis, since the basis will be the same as *backup.sh*.

This script can be used for both automatic and interactive backups.



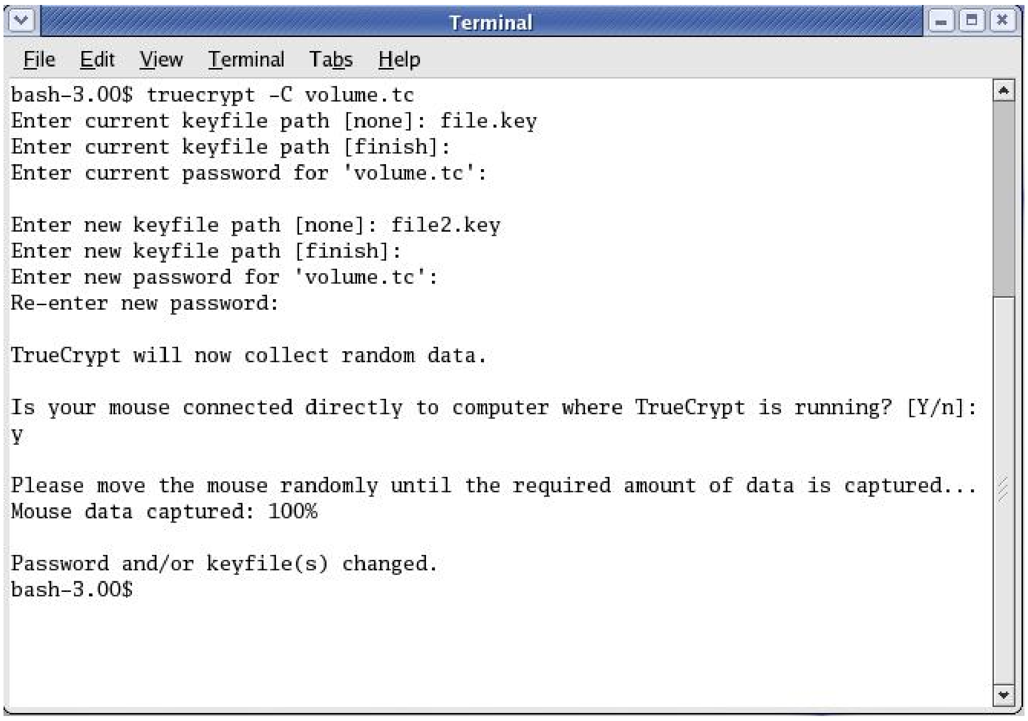
```
bash-3.00$ ./backup.sh
Username:
obuser
Directory to syncronize:
/user/sync
Checking if volume is mounted...
Mounting volume...
Synchronizing...
building file list ... done
sync/
sync/file1
sync/file2
sync/file3
sync/file4
sync/file5
sync/file6

sent 442 bytes  received 140 bytes  388.00 bytes/sec
total size is 30  speedup is 0.05
Unmounting volume...
bash-3.00$
```

Figure 10: *backup.sh* without parameters



```
bash-3.00$ ./backup.sh obuser /user/sync
Checking if volume is mounted...
Mounting volume...
Synchronizing...
building file list ... done
sync/
sync/file1
sync/file2
sync/file3
sync/file4
sync/file5
sync/file6

sent 442 bytes  received 140 bytes  1164.00 bytes/sec
total size is 30  speedup is 0.05
Unmounting volume...
bash-3.00$
```

Figure 11: *backup.sh* with parameters



```
bash-3.00$ truecrypt -C volume.tc
Enter current keyfile path [none]: file.key
Enter current keyfile path [finish]:
Enter current password for 'volume.tc':

Enter new keyfile path [none]: file2.key
Enter new keyfile path [finish]:
Enter new password for 'volume.tc':
Re-enter new password:

TrueCrypt will now collect random data.

Is your mouse connected directly to computer where TrueCrypt is running? [Y/n]:
y

Please move the mouse randomly until the required amount of data is captured...
Mouse data captured: 100%

Password and/or keyfile(s) changed.
bash-3.00$
```

Figure 12: Changing the keyfile and password of a TrueCrypt volume

changekey.sh

It is desirable to change the keyfile with regular intervals. If this is done manually, it is necessary to first create a new keyfile, and then run the correct command to change the keyfile. Figure 12 shows the change of keyfile using the TrueCrypt command. *changekey.sh* does this automatically, by collecting the name of the volume from the previously mentioned configuration file, and thereby deriving the name of the keyfile. It then creates a new keyfile, changes which keyfile is associated with the volume, and renames this to the same as the original keyfile. In this way *mount.sh*, and other relevant scripts, still refer to the correct path. The old keyfile is deleted, as it is no longer of any use. As when creating volumes, the random information source is set to be /dev/random and /dev/urandom.

This script requires some user interaction, as the TrueCrypt command has no option for providing the new password (which in this situation will be empty). The user therefore needs to press enter twice when running this script. This issue is discussed in chapter 10.

restoreheader.sh

This script automates the restoring of the volume header, to the original header stored when the volume was created. To restore a header, the user needs to know the correct command and syntax, as well as the volume name and the path to the header file. *restoreheader.sh* automates this action, by reading the information about the volume from the configuration file, generating the name of the header file based on the volume name and restoring the header.

This script can be used as a part of the disaster recovery, and also in the event that the keyfile stored in AFS is lost, to regain access to the volume.

6.2 Scripts used in the version in production at CERN

As stated in chapter 5, the IT staff at CERN decided that at this point, it is not necessary with one volume for each user, and not necessary to unmount the volume after each use. Therefore, the basis for the script development is different. It is still desirable with a script to create a volume, and also a script for mounting. In this situation too, it is desirable with a script to create a random password, a script to check if a volume is mounted and a script to restore the header of a volume. These scripts are the same as those presented above, and will therefore not be covered here. Finally, a script used to change the password of a volume is presented.

createC.sh

The main differences between this script and *createvol.sh* is that here the volume is not changed to accept keyfiles instead of a password, and that the file format is set to ext3. At CERN there will only be offsite backup system administrators, people who can log in as root on the offsite backup server, who will create new volumes. Therefore, it is no problem to format these to ext3. However, this requires knowledge about the necessary actions to do this, among other things it is necessary to export an environment variable to prevent the system from freezing during the formatting. There are some issues related to this point which are discussed in chapter 9.

createC.sh performs four actions:

- Create new volume using password from *random.sh*
- Format volume
- Backup volume header
- Create configuration file
- Mount volume

It is then the responsibility of the person creating the volume to store the password according to the key management policy presented in chapter 7. Also, the different user spaces on the volume must be created and the correct permissions must be set.

mountC.sh

This does the same as the script *mount.sh* presented above, except checking the remaining capacity of the volume. The only difference is that instead of reading a keyfile, the user is prompted for a password. This password is created by the use of *random.sh* either when the volume was created, or if the password has been changed since the creation of the volume.

backupC.sh

This script is fairly similar to *backup.sh*, except that it only checks if the volume is mounted before running the backup, instead of mounting it and unmounting when the backup is done. If the volume is not mounted when the request is made, an e-mail is sent to the offsite backup administrators, the user is notified, and the backup is aborted.

Scripts for automated backups running during the night may in principle remain unchanged, since there is no need to mount and unmount the volume every time. However, it is also in this case wise to check if the volume is mounted before running the backup, and to send an alert and abort the backup if the volume is not available. The structure of this will be very much like *backupC.sh*.

changepass.sh

As with the change of keyfile in the previous section, it is desirable to be able to change the password in this system design. This script consists of the actions to create a new password, using *random.sh*, and to change the password of the volume, using the previously described TrueCrypt command. The process of changing password is like the one presented in figure 12, except for the options provided. The user has to perform the action of manually entering or copying the new password when prompted. He is also responsible for the printing and secure storage of this password.

7 Policy – key management and administration

This chapter will present the key management policy corresponding to both the system design presented in section 5.1, and the version presented in section 5.2. First, there will be a clarification of what is necessary to take into consideration when designing key management policies for a system like this. Next, the key management policy itself is presented. The chapter is completed with a description of the necessary actions to take with regards to a disaster recovery plan, and what actions should be taken in the event of an actual disaster.

Information specific for CERN, such as e-mail addresses, computer names and file paths are not included.

For a system like the one presented in this thesis, a policy regarding the key management – key generation, key distribution etc. – and guidelines to administrators and users of the backup system, as well as policies on how the backups should be recovered in the case of an disaster is important aspects.

The aspects that are addressed in this chapter are:

- Who is allowed to generate new volumes and keys?
- What is the process of generating new volumes and keys?
- How is the generated keyfile/password distributed and stored?
- Who is allowed to access the volumes and keyfile/password?
- How should the offsite backup service be used?
- How is disaster recovery handled?
  - Where is the password and header file stored?
  - Who is allowed access to the password?
  - How is the data restored?
- How should it be handled when a user leaves the organization?
- Who is responsible for testing the disaster recovery plan?
- Who is responsible for the maintenance of the policy?

7.1 Policy for the secure offsite backup service

This policy clarifies the use of the secure offsite backup service. The service is available to give the users the opportunity to secure a live copy of their data to be able to recover from a prospective disaster which renders the data in ordinary storage unavailable.

Creation of volume

All users have permission to create new volumes. However, when a new volume is created, the old one will not be accessible for script-based backups any more. The scripts will mount the volume represented in the current configuration file corresponding to the service account user name.

It is desirable that an offsite backup administrator is contacted before a new volume is created. Information about who the offsite administrators are can be obtained by contacting the relevant IT section.

New volumes are created by using the script *createvol.sh*, available in the folder ~/private of the relevant service account. The scripts are also available in CVS.

If the scripts for some reason are unavailable, the volume should be created by carefully following the directions found in the documentation.

If the volume is created using the provided script, a strong password is generated automatically. However, if the volume is created manually, the password should be generated before the creation of the volume is started. This can be done using the script *random.sh*, or some equivalent method. The password must be written down/printed, put in an envelope marked with the service account name and transported to the secure location. When running *create.sh*, the user is prompted to do this, and confirm that it is done.

NOTE: If the password is not written down and transported to the secure location, the data will be lost in the case of a disaster.

The keyfile needed to mount the volume is generated by TrueCrypt, and stored in the AFS ~/private folder of the service account.

During the creation of the volume, the volume header is also backed up, one copy is stored on the offsite backup machine and one is stored in AFS.

Mounting and unmounting volumes

The volume can be mounted and unmounted by a user logged in to the remote backup server using a valid service account.

Mounting is done by using the script *mount.sh*, which is available in the ~/private folder of the relevant service account. If the script is not available, the volume must be mounted manually. The procedure of a manual mount is described in the documentation. Necessary information is volume name, keyfile path and mount point.

Unmounting is done by using the script *umount.sh*, which is available in the ~/private folder of the service account. If the script is not available, the volume must be unmounted manually. The procedure of a manual unmount is described in the documentation. It is very important to pay attention to the syntax of the command, and to control that the volume was unmounted.

Taking backups

Backups using rsync are taken using the script *backup.sh*, available in CVS. This script is run on the local client machine, and automatically mounts and unmounts the required volume on the offsite backup server.

To manually take a backup, it is necessary first to logon to the offsite backup server, mount the volume, transfer the data and unmount the volume. This process is described in the documentation. To ensure the security of the data, it is very important to unmount the volume before closing the connection to the remote backup server.

Accessibility

The volumes are accessible to those who hold a valid service account user name and are able to connect to the offsite backup server by the use of SSH. The service account must also correspond to the permissions set on the volume.

The passwords (one for each volume) located in the offsite secure location is only accessible by offsite backup administrators. A full disaster recovery of the volumes is therefore only possible for these individuals. In the event of a disaster, the users should contact the relevant IT section to enable the recovery of their volumes.

Disaster recovery

In the event of a disaster, where the keyfile stored in AFS has been destroyed, the password stored in the secure location must be retrieved. This must be done by an offsite backup administrator. The script *restoreheader.sh* is used to restore the header file stored on the backup server. The volume must then be mounted manually, providing the original password when prompted. After this, assuming that AFS is again available, the volume must be changed from the use of password to the use of keyfile. The password must then be returned to the secure location.

Testing

The disaster recovery plan should be tested to see that the process is feasible and correctly adapted to the users' needs. This should be done by the offsite backup administrators, in cooperation with the persons responsible for each service account's backups and the person(s) who control the access to the secure safe. The launching of the test is the responsibility of the offsite backup administrators.

Testing should be done regularly, to see that the requirements to the service is still satisfied, and to ensure that the different contributors know their tasks.

Keyfile revocation

The keyfile should be changed with regular intervals, at least once every six months.

In addition to this, the keyfile should be changed when an individual who has had access to it should no longer have this access. Reasons for this may be that the person leaves the organization or that his/her responsibilities change.

Responsibility for policy maintenance

The offsite backup administrators are responsible for keeping this policy up to date. However, other individuals using the service must report any shortcomings or mistakes in the policy.

Possible threats and how they are countered

Eavesdropping on transmission:
The data is transferred using SSH, and thereby encrypted.

Malicious insiders
The data is stored in encrypted form on the backup server, and is only accessible to those holding a valid Kerberos ticket for the service account. The data is also secure against an attacker who gains root (administrator) access to the server.

Physical attacks
The data is not available in cleartext without the correct keyfile. The keyfile is not stored on the backup server. The data is secured against a physical attack on the server (e.g. theft), since the data will be inaccessible.

7.2 Policy for the version in production at CERN

Creation of volume

Only offsite backup administrators are allowed to create new volumes and generate new keyfiles. Information about who are the offsite administrators can be obtained by contacting the relevant IT section. The creation process requires root-access to the relevant backup server.

New volumes are created with the script *createC.sh*, available in the folder ~/private of the *obadmin* service account. The scripts are also available in CVS.

If the script for some reason is unavailable, the volume should be created by carefully following the directions found in the TWiki documentation page.

If the volume is created using the provided script, a strong password is generated automatically. However, if the volume is created manually, the password should be generated before the creation of the volume is started. This can be done using the script *random.sh*, or some equivalent method. After creating the volume, the password must be written down or printed in 5 – five – copies. Four copies are given to trusted personnel, preferably offsite backup administrators, to keep secure. These must be kept away from the CERN Meyrin site. One copy should be kept securely on the site. This can e.g. be done by locking a note in a drawer, or by keeping the password in an encrypted file on the desktop computer of one of the administrators. This copy should be available if the offsite backup server needs to be rebooted.

If there shall be different user spaces in the volume, these must be created when the volume is created, and the correct permissions must be set: the service account must be set as owner of the directory, and the permissions should be set to 0600; the owner, and no one else, is able to read and write, and no one is allowed to execute code.

Mounting the volume

If the server is rebooted, or if the volume for some other reason has been unmouted, there is a script – *mountC.sh* – which should be used to remount the volume. Instructions on how to manually mount a volume is found in the TWiki documentation page. Necessary information is volume name and mount point, and the correct password must be provided for the mounting to be successful.

Accessibility

The volume is accessible to those who hold a valid service account user name and password, and can connect to the offsite backup server by the use of SSH.

Disaster recovery

In the event of a disaster, where the onsite copy of the password has been destroyed, one of the copies of the original password held by the administrators must be obtained. If the password has been changed since the volume creation, the script *restoreheader.sh* is used to restore the header file. *mountC.sh* can then be used to remount the volume, providing the original password when prompted.

Testing

The disaster recovery plan should be tested to see that the process is feasible and correctly adapted to the users' needs. This should be done by the offsite backup administrators, in cooperation with the persons responsible for each service account's backups. The launching of the test is the responsibility of the offsite backup administrators.

Testing should be done regularly, to see that the requirements to the service are still satisfied, and to ensure that the different contributors know their tasks.

Password revocation

The system is based on "lacy revocation": The password must be changed when someone who has had access to the password for some reason should not have this access any more. Reasons for this may be that the person is leaving the organization, or that his/her responsibilities have changed.

In addition to this, the password should be changed every six months. When the password is changed, the onsite copy of the old password must be destroyed.

Responsibility for policy maintenance

The offsite backup administrators are responsible for keeping this policy up to date. However, other individuals using the service must report any shortcomings or mistakes in the policy.

Possible threats and how they are countered

Eavesdropping on transmission

The data is transferred using SSH, and thereby encrypted.

Malicious insiders

The data stored on the offsite backup server is protected by the standard Linux file permissions. This is the same level of protection as used where the data is otherwise stored at CERN, and is regarded as safe enough.

Physical, off-line attacks

The data is not available in clear text without the correct password. This password is not stored on the backup server. The data is secured against a physical, off-line, attack on the server (e.g. theft), since the data will be encrypted, and thereby inaccessible, when the machine is rebooted.

8 Procedures

This chapter presents the procedures necessary to carry out the policies presented in the previous chapter.

It must be clearly defined who holds the different responsibilities related to the offsite backup:

- Who are the offsite backup administrators?
- Who is responsible for the creation of new service accounts and new volumes?
- Who is responsible for the access to the safe in the secure location?
- What are the procedures for the disaster recovery process, and who is responsible for testing them and carrying them through in the event of a disaster?

The offsite backup administrators are chosen members of the IT-DES-SIS section at CERN. In the system design put in production at CERN (see section 5.2), these individuals have the responsibility of creating new volumes, and all tasks related to this, and to request the creation of new service accounts from the user registration office.

In addition to this, these, especially the one who creates a new volume, are responsible for the proper handling of the new password. This must be printed or written down without any risk of exposure, in the correct number of copies, and given to the chosen individuals to be kept securely off the Meyrin site. These copies must as soon as possible, preferably immediately, be transported to their new locations. Also, one copy should be kept on the Meyrin site, secure but available when needed. This would be in the case of a reboot of the offsite backup machine, which would require the volume to be remounted. This is also the responsibility of the offsite backup administrators, and should be done using the dedicated script.

In the complete system design, the offsite backup administrators are responsible for creating new volumes and requesting the creation of new service accounts. When a new volume is created, the individual creating it is responsible for transporting the password to the secure location safe. There should be a limited number of people having access to the safe, and one of these must be contacted to be able to deposit the password. The password should be put in a sealed envelope, and marked with the name of the volume. Information about which persons are allowed to retrieve the envelope(s) holding volume passwords must be clearly stated to the individuals who have access to the safe.

For each group or service holding a service account, there must be one person appointed as responsible, and a substitute in case the responsible person is not available. These persons must be trained to handle the different problems that may arise during use of the system. Testing of the backup/restore functionality is an important part of this training. When there is a problem with the backups, e.g. if the volume can not be mounted or unmounted, these persons are notified, as well as the offsite backup administrators.

All users should also be given instructions and training on how to use the system: How to make a backup manually and how to manually mount and unmount the volume to manage the data stored on the offsite backup server.

When a person having access to the keyfile of a volume should no longer have this access, the keyfile must be changed. If this person at some point had access to the password stored in the secure location, to be used for disaster recovery, this password must also be changed, and the volume header backed up again.

Similarly, in the system used at CERN, the current password of the volume must be changed when a person having had access to the onsite copy of the password should not have this access anymore. If this person has also had access to one of the offsite copies of the password, this password must also be changed, and the volume header backed up again.

It is important that the disaster recovery plan is clearly defined and thoroughly tested. This way, a disaster will be handled more efficiently, since different actions and responsibilities are known beforehand. In the case of the offsite backup system at CERN, the responsibility of the offsite administrators will be to recover the password and remount the volume, if the backup server is shut down as a result of the disaster. After that, or if the offsite backup server is not shut down (i.e. the data is still accessible), the responsibility of recovery from the data stored on the backup server lies on the different users and should be a part of the different group's disaster recovery plan.

In the complete system design, the offsite backup administrators will, on the request from the different users, need to travel to the secure location to retrieve the different passwords from the safe. After this, the different headers must be restored and the volumes mounted using the correct passwords. The different users must then restore their data, in accordance with their group's disaster recovery plan. The offsite backup system presented in this thesis will not function properly before AFS is running again, and in the meantime, the password will be needed to mount the volume if it is necessary to take new backups. Depending on the length of this time period, and the amount of people handling the password, it may be desirable to change the password and take a new backup of the volume header when AFS is functioning again, before changing the volume to accept keyfiles again. The current password must then be transported to the secure safe, following the same procedure as with the initial password.

There must be a defined line of responsibility in the event of a disaster. It must be clear who have the permission and responsibility of retrieving the key(s) from the secure location if all offsite backup administrators are unavailable.

9 Discussion

In this chapter, the previously presented system design is discussed. The discussion will first consider the complete system design presented in section 5.1, with regards to user inconvenience and overall security. Afterwards, the version put in production at CERN is discussed (presented in section 5.2), considering the alterations made to the complete system design to adapt it to the current situation at CERN.

It is often a challenge to keep the user inconvenience to a minimum when developing a secure system. The introduction of security to a system often increases the inconvenience experienced by the users. The system presented in this thesis makes the action of taking encrypted backups totally transparent to authorized users. The volumes are automatically mounted and unmounted when a user runs the backup script and thereby logs into the remote backup server using SSH. The data is encrypted on-the-fly when transmitted. The only noticeable change for the users, except of the change of command used when running the backup, is that the transmission to a file container volume are slower than a transmission to unencrypted space on the disk. The consequence of this will differ depending on the amount of data transmitted. Table 1 shows the time differences when transmitting 5GB of data using scp and rsync. When transmitting to a partition container volume, there was no significant difference in the transmission time.

	plain text	encrypted	%
rsync	$\approx 11min$	$\approx 13min$	+18.2%
scp	$\approx 9:15min$	$\approx 12:50min$	+36.6%

Table 1: Time spent transmitting 5GB of data

The users can still use SSH to connect to the offsite backup server as before to manage the backed up data. The only additional action needed by the user is to run *mount.sh* to make the data available, and to run *umount.sh* when finished, before logging out. It is, however, crucial that the volume is unmounted, since a mounted volume would be available to anyone holding the right permissions, e.g. an adversary that has acquired root access to the machine. As mentioned in section 3.1, it will in this kind of a situation be necessary for the user to know what to do and why it has to be done, as well as to know how to do it. Proper training and information must be given to ensure this. Furthermore, it is desirable with a function that unmounts a volume after a certain time of no activity, to decrease this vulnerability.

As long as the volumes are properly unmounted after use, the data will be protected. To access the keyfile necessary to mount a volume, a valid Kerberos ticket is needed. To get this ticket, the user must first be authorized to log in to the remote machine using SSH with cryptographic keys. In this way, it is never necessary to enter a password. This security is also dependent on the users' ability to remember to lock their screens when leaving their desks. If an adversary is able to get access to a computer where an authorized user is logged in, this gives him the opportunity to connect to the remote backup server and mount the volume, and thereby have access to the data. One way to handle this vulnerability is to use only a password, or both a password and a keyfile, to mount the volume. This is discussed further in chapter 10.

Another benefit of storing the keyfile in AFS is that even though an adversary gains root access to the remote backup machine, he will still not be able to read the keyfiles necessary to mount the volumes, because he will not have the correct Kerberos credentials. As a result of this, the data stored in the encrypted volumes will still be secure.

The volumes and keyfiles are created using /dev/random and /dev/urandom as random input instead of the default mouse movement or keystrokes. The reasons for this are stated in chapter 6. The use of mouse movements is not possible when TrueCrypt is run on the remote server, which only would leave the option of keystrokes. This introduces a substantial increase of user inconvenience, since the entering of 320 "random" characters is quite troublesome, and the actual randomness of the characters is very questionable. According to the /dev/random Linux manual page [32], the output "should be suitable for uses that need very high quality randomness". Some sources, such as [57, 58], do however state that this is not always the case. Nevertheless, the randomness of /dev/random and /dev/urandom is in this case regarded as good enough and also more ensured than the randomness of the user entering keystrokes.

The reason why /dev/random is used as the source when creating volumes and /dev/urandom is used when creating keyfiles, is simply because of efficiency. When the pool is already used to provide 320 bytes, it will take some time before there are 320 new bytes available and controlled by /dev/random. /dev/urandom will use these bytes without controlling the entropy, and therefore without delay. One may argue that when using several hours to create a volume, a few seconds extra for the keyfile will not make a difference. It is, however, also a case of giving the user proper feedback, when the software waits for input from /dev/random, it may seem like the process have stalled, and the user may try to abort and start over. By using /dev/urandom, this is avoided.

The keyfile created using TrueCrypt's *——keyfile-create* function is a 64-bit binary file, randomly generated. This gives $2^{64} \approx 1.8 * 10^{19}$ different possibilities. The script *random.sh* generates a password consisting of 20 characters from a field of 72. This gives $72^{20} \approx 1,4 * 10^{37}$ possible results. When considering a brute force attack, where all possibilities are attempted to break the password/keyfile, and taking into the consideration that the same data is stored unencrypted elsewhere at CERN, the strength of the keyfile and password is regarded as very good. The users are also requested to change the keyfile or password approximately every six months. In addition to this, the strong password generated when creating the volume ensures that an adversary is not able to use the backed up header file stored on the server to access the data.

Another issue concerning key lengths is the encryption algorithm used for the traffic encryption between AFS servers and the client (in this case the offsite backup server). As mentioned in section 4.2, this algorithm uses a rather short key length, and is therefore vulnerable of attacks. Since this is an integrated part of AFS, it is difficult to change it before it is changed in the AFS system. Changing the keyfiles on a regular basis is important to minimize this vulnerability.

It might be argued that this system design puts too much confidence in the users. They are able to create their own volumes, and also to change keyfiles of the existing volume. However, by using the provided scripts, these actions should be possible without problems. If the keyfile of a volume is changed, this simply replaces the original one, and thereby ensures that the volume can still be mounted. The main problem is where user interaction is necessary, especially where the user needs to press enter twice to set the new password. If the user does anything but press enter in this situation, the volume will need the password to be mounted, and the system will not function as intended. This issue is also discussed further in chapter 10.

Another vulnerability that is a result of the freedom of the users is that the backed up header file is available to anybody logged in with the correct service account. This makes it possible for an insider to corrupt the header file, and thereby make the disaster recovery impossible. It may be desirable to store the header file in the safe location with the password, or to alert an offsite backup administrator when a new volume is created, so that the header file can be copied to a location on the backup server only accessible by root users.

The IT department at CERN has, in their contingency planning, decided that the CERN site in Prevessin is sufficiently far away to be the location of the offsite backup facilities. The distance between the Meyrin site and the Prevessin site is about three kilometers. On the one hand, this rather short distance simplifies the process of recovery in the event of a disaster, since the necessary information and equipment is close at hand. On the other hand, it increases the possibility of both sites being affected e.g. if a severe natural disaster should occur. This is, however, a decision that was made before the start of this project, and it will most likely never be a problem that the two sites are not further apart.

As mentioned, the requirements to the system at CERN changed during the progress of this project. It was decided that there was only need for one large volume, and that the volume did not need to be unmounted after each use. These alterations affect the overall security of the system. That is also the case when concerning the decision of keeping the password for disaster recovery in the possession of the different offsite backup administrators. These issues will be discussed next.

The fact that the volume remains mounted at all times leaves it vulnerable to insiders and other adversaries somehow getting access to the machine without cutting the power. This is, however, not seen as too big a risk, because all the data stored on this server is stored in cleartext other places at CERN, and are therefore equally vulnerable there.

An important issue about the system put in production at CERN is the management of the password needed for disaster recovery. Since it was not feasible to find a secure safe or other location to store the password in, it was decided that each offsite backup administrator would bring one copy of the password home with him, and store it securely there. First of all, this opens for insecure storage of the password, since it is not easy to monitor that the different persons stores the password according to the policies and procedures. It may also be possible that one or more of the copies are lost, since they are not stored in a controlled environment.

Temporarily, while the system is put in production, the password is also stored in a text file in the *obadmin* AFS account. To connect via SSH to this service account on the backup server, a password is needed. Without this password, it is not possible to log in to the server, and the proper Kerberos tickets are not obtained. The text file holding the password is then not accessible. This solution can also be used as the way of keeping the password available on site, maybe combined with OpenSSL or something similar to encrypt the file, with a password that is easier to remember. It may be argued that protecting a strong password with a weaker password is the first step in weakening the total security, and that it would be better to use a smart card, biometrics or other authentication means. However, the file holding this password will be protected by three other passwords: First, it is necessary for the user to log in to a local computer. Then, he/she must log in to the remote backup server using SSH. And at last, it is necessary with a password to access the file. This can in most cases be assessed to be as secure as locking a note in a drawer.

In extreme situations, personnel may be rendered unavailable. If this includes the individuals keeping the password in their homes, and the password stored onsite is also unavailable, the result may be that the volume can not be mounted. Although this is unlikely to happen, it should be considered. Since the situation at CERN does not enable the storage of the password securely, this risk must be taken in this case.

This key management solution is chosen by those responsible at CERN as the most feasible solution.

Finally, there are some general aspects of the system design that should be commented, especially concerning the adaptation of the system design to other situations.

In the complete system design, the volume is formatted to FAT. However, in many situations, it may be desirable to use other file formats. In those cases, it is necessary to have administrator rights – root access in Linux – on the computer where the volume is created. In the version of the system design put in production at CERN, this was possible because offsite backup administrators would be the ones creating the volumes. The complete system design makes it possible for users to create new volumes for themselves. Obviously, it is not desirable to give root access to all the users.

Also, if an administrator logs in as root to create a new volume, the corresponding keyfile should be stored in the AFS account of the correct service account. Since the root user does not have the correct Kerberos ticket, he will not have permission to store the keyfile there.

Both these issues can be solved by giving the service accounts the permission to use the *sudo*-command; to be able to perform one single command with administrator privileges. It can be specified which functions should be allowed, and if the user should enter a password or not to do this.

A final disadvantage of the system is that if it is desirable to re-encrypt the volumes with a different encryption algorithm, or use another hash algorithm to encrypt the volume header, a new volume must be created and all the data must be transferred to the new volume. In an environment where large amounts of data is held, this would both leave the data inaccessible for the time it takes to transfer it, and it would not be possible to take new backups while the process is in progress.

10 Further Work

This thesis presents a secure offsite backup system utilizing AFS with Kerberos, SSH and TrueCrypt. There are some areas where the design may still be improved. These areas are presented in this chapter.

- To secure the keyfile better, it may be desirable to encrypt it when stored in AFS, e.g. by the use of OpenSSL, with a password to decrypt it when needed. This would remove the vulnerability of an adversary getting access to a computer where an authorized user is already logged in. It is also possible to require both a keyfile and a password when mounting a volume. This would serve the same purpose.
- It may be desirable to develop a solution to transfer the keyfile in encrypted form, without using the traffic encryption of AFS.
- In the version of TrueCrypt released since this project was commenced, the possibility for non-root users to mount volumes is removed, because it was detected that there was a possibility of denial of service attacks, since there is no check of where the volumes are mounted. This can be solved by giving the users the permission to run the *sudo*-command. The commands allowed can be specified, so volume name and mount point may be added to the permission, to make it impossible for a user to mount the volume anywhere else.
- There should be a function controlling the use of mounted volumes. If there has been inactivity for a given amount of time, e.g. an hour, it can be assumed that the user has forgotten to unmount the volume when logging out, and the volume should be unmounted.
- At this point, the backup script is only made for Linux machines. It is desirable with scripts that can also be used on a computer running Microsoft Windows.
- For small organizations, it is possible to install TrueCrypt on each client computer, and use e.g. SSHFS [59] to mount a volume stored on the remote server to a directory on the local client machine. The data on the volume would then be transferred in encrypted form, and decrypted in the memory of the client. In organizations not using AFS, the keyfile could also be stored on a smart card, protected by a pin code or password.
- The system presented in this thesis only has a command line user interface. It would be desirable to develop a graphical user interface, e.g. a web interface. This may for instance be done using PHP, where the different scripts presented in this thesis can be run from PHP using the *shell_exec*-function.
  Kerberos supports authentication delegation, or authentication forwarding. This means that an intermediary machine, such as a web server, is able to access remote resources on behalf of the user [60]. This can be used in a system such as this: By the use of a web interface, user A can forward his/her Kerberos ticket to the web server, which

is then able to authenticate to the remote backup server. This way, the system would still have the Kerberos authentication which much of the security in this system design is based on, in addition to a user interface which is comprehensive also to users who are not comfortable with command line-based user interfaces.

- In the script *createkey.sh*, the user is prompted for password. In the solution presented in this thesis, the user then needs to press enter twice. By using an “expect” script, it is possible to avoid this. This script are configured to “listen” for specific requests, and then respond to this request with a predefined value.

11 Conclusions

This thesis presents a secure offsite backup system that can be used in large organizations, combining the AFS distributed file system, using Kerberos for authentication, with TrueCrypt for encryption. The system design presented in section 5.1 is the one best suited for the purpose, but the version put in production at CERN also shows that there is several ways to change the system to adapt it to different requirements.

It is shown that software tools such as TrueCrypt can be successfully used for system designs like the one described in this thesis. Combined with AFS, Kerberos authentication and the use of SSH it can (in most situations) be part of a system that provides an organization with a secure offsite backup to be incorporated in a disaster recovery plan.

One of the main goals of this project was to design and develop a secure offsite backup system that would be as transparent as possible to the users. This is done by developing scripts to carry out the different functions; mount/unmount, make backups and recover in the event of a disaster. To minimize the necessary user interaction, the system utilizes SSH using cryptographic keys and the TrueCrypt keyfile functionality. This gives total transparency to the user when making backups.

Some user interaction is needed when creating new volumes, changing keyfiles and recovering from a disaster. However, these are actions where some need for interaction is expected, and the scripts presented in this thesis reduce the necessary interaction substantially.

The main requirements to the security of the offsite backup system, presented in section 4.1 was:

- The data must be transferred securely.
- The data stored by one client must not be visible by any other client.

In addition to this, it was important to ensure the security of the data if an adversary gains physical access to the server.

All these requirements are satisfied in the system design presented here, and the security is therefore assessed to be satisfactory.

The system design presented in this thesis may be used in different kinds of organizations, with different requirements to security. One variation of the system are presented in this thesis, while suggestions to other adaptations are presented in chapter 10. This opens many possibilities for the use of the system.

Bibliography

[1] What are CERN's greatest acievements? Nobel Prizes. Available from http://public.web.cern.ch/public/Content/Chapters/AboutCERN/Achievements/NobelPrizes/NobelPrizes-en.html Last visited 25.10.07.

[2] All about CERN... in 7 questions! Available from http://press.web.cern.ch/public/Content/Chapters/AboutCERN/AboutCERN-en.html Last visited 31.10.2007.

[3] Ylonen, T. & Lonvick, C. jan 2006. The Secure Shell (SSH) Protocol Architecture. Available from ftp://ftp.rfc-editor.org/in-notes/rfc4251.txt Last visited 25.10.2007.

[4] Bauer, M. 2003. Paranoid penguin: rsync, part i. *Linux J.*, 2003(107), 10.

[5] Bauer, M. 2003. Paranoid penguin: rsync, part ii. *Linux J.*, 2003(108), 13.

[6] Smith, T. 2004. Using scp (secure copy) to transfer files. *Digital Library of Information Science and Technology*. Available from http://dlist.sir.arizona.edu/326/Using%5Fscp.html Last visited 25.10.2007.

[7] Robey, D., Welke, R., & Turk, D. 2001. Traditional, iterative, and component-based development: A social analysis of software development paradigms. *Inf. Tech. and Management*, 2(1), 53–70.

[8] Storer, M., Greenan, K., & Miller, E. 2006. Long-term threats to secure archives. In *StorageSS '06: Proceedings of the second ACM workshop on Storage security and survivability*, 9–16, New York, NY, USA. ACM Press.

[9] F. Graf, S. W. nov 2005. A capability-based transparent cryptographic file system. In *Proceedings of the 2005 International Conference on Cyberworlds (CW'05)*.

[10] Riedel, E., Kallahalla, M., & Swaminathan, R. 2002. A framework for evaluating storage system security. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA. USENIX Association.

[11] Whitten, A. & Tygar, J. 2005. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, 679–702. O'Reilly.

[12] Whitten, A. & Tygar, J. dec 1998. Usability of security: A case study. *Technical Report CMU-CS-98-155, Carnegie Mellon University*.

[13] Blaze, M. 1993. A cryptographic file system for UNIX. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, 9–16, New York, NY, USA. ACM Press.

[14] Hughes, J., Feist, C., Hawkinson, S., Perraut, J., O'Keefe, M., & Corcoran, D. 1999. A Universal Access, Smart-Card-Based, Secure File System. *Atlanta Linux Showcase*.

[15] Adya, A., Bolosky, W., Castro, M., Cermak, G., Chaiken, R., Douceur, J., Howell, J., Lorch, J., Theimer, M., & Wattenhofer, R. December 2002. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *5th Symposium on Operating Systems Design and Implementation (OSDI), Boston, Massachusetts*.

[16] Cattaneo, G., Catuogno, L., Sorbo, A. D., & Persiano, P. 2001. The design and implementation of a transparent cryptographic file system for UNIX. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, 199–212, Berkeley, CA, USA. USENIX Association.

[17] OpenSSL: The open source toolkit for SSL/TLS. http://www.openssl.org/ Last visited 25.10.2007.

[18] Thompson, K. 2007. Backup encryption. *Sys Admin*, 16(3), 12–17.

[19] The GNU Privacy Guard - GnuPG.org. http://www.gnupg.org/ Last visisted 25.10.2007.

[20] Callas, J., Donnerhacke, L., & H. Finney, R. T. nov 1998. RFC2440: OpenPGP message format. Available from ftp://ftp.rfc-editor.org/in-notes/rfc2440.txt Last visited 25.10.2007.

[21] da Silva, J. & Guthmundsson, O. 1993. The Amanda network backup manager. In *LISA '93: Proceedings of the 7th USENIX conference on System administration*, 171–182, Berkeley, CA, USA. USENIX Association.

[22] Garcia, L. & Pragin, P. *Secure Network Backups in a Heterogeneous Environment in the Time it Takes to Have Pizza Delivered (All Using Open Source Software!)*. Available from http://amanda.zmanda.com/quick-backup-setup.html Last visited 25.10.2007.

[23] Amanda: Open source backup. http://amanda.zmanda.com/ Last visited 25.10.2007.

[24] Zmanda, Inc. *Open source backup software Amanda*. Available from http://network.zmanda.com/pdf/amanda-whitepaper.pdf Last visited 25.10.2007.

[25] Zmanda, Inc. *The Open Source Backup Wiki*. http://wiki.zmanda.com/index.php/Main_Page Last visited 25.10.2007.

[26] Windows client. http://wiki.zmanda.com/index.php?title=Windows_client Last visited 25.10.2007.

[27] Howto use cryptsetup with luks support. Available from http://feraga.com/node/51 Last visited 24.10.2007.

[28] Security products from Jetico. http://www.jetico.com/ Last visited 25.10.2007.

[29] TrueCrypt. http://www.truecrypt.org/ Last visited 25.10.2007.

[30] Doernhoefer, M. sept 2006. Surfing the net for software engineering notes. *ACM SIGSOFT Software Engineering Notes*, 31(5), 6–15.

[31] Bauer, M. 2002. Paranoid penguin: BestCrypt: cross-platform filesystem encryption. *Linux J.*, 2002(98), 9.

[32] Manual reference pages - RANDOM (4). Available from http://www.squarebox.co.uk/cgisquarebox/manServer/usr/share/man/man4/random.4 Last visited 25.10.2007.

[33] BestCrypt source code. Available from http://www.jetico.com.

[34] RSA Laboratories, R. D. S. I. March 1999. PKCS #5 v2.0: Password-Based Cryptography Standard. Available at ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf Last visited 25.10.07.

[35] Ferguson, N. & Schneier, B. 2003. *Practical Cryptography*. Wiley Publishing, Inc, Indianapolis, IN, USA.

[36] Information Security Forum. *The standard of good practice for information security*, 2005. Available from http://www.isfsecuritystandard.com/SOGP07/pdfs/SOGP_2007.pdf Last visited 25.10.2007.

[37] Bowden, J. feb 2003. Security policy. What it is and why - The Basics. Available at https://www2.sans.org/reading_room/whitepapers/policyissues/488.php Last visited 25.10.07.

[38] Wills, L. dec 2002. Security policies: Where to begin. Available from http://www.sans.org/reading_room/whitepapers/policyissues/919.php Last visited 25.10.2007.

[39] Höhe, K. & Eloff, J. 2002. What makes an effective information security policy? *Network Security*, 14–16.

[40] Menezes, A., Vanstone, S., & Oorschot, P. V. 1996. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA.

[41] Schneier, B. 1996. *Applied Cryptography. Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA.

[42] Schäfer, G. 2003. *Security in fixed and wireless networks - an introduction to securing data communications*. Wiley Publishing, Inc., West Sussex, England.

[43] Merkow, M. & Breithaupt, J. 2007. *Information Security: Principles and Practices*. Pearson Education, Upper Saddle River, NJ, USA.

[44] Fallara, P. 2004. Disaster recovery planning. *IEEE Potentials*, 22(5), 42–44.

[45] OffSiteBackup < DESgroup < TWiki. Available from https://twiki.cern.ch/twiki/bin/viewauth/DESgroup/OffSiteBackup Last visited 31.10.2007.

[46] Scientific Linux CERN 4 (SLC4). Available from http://linux.web.cern.ch/linux/scientific4/ Last visited 31.10.2007.

[47] OpenAFS. Available from http://www.openafs.org/main.html Last visited 31.10.2007.

[48] Mullender, S., ed. *Distributed Systems*, chapter 9. Addison Wesley Publishing Company, 1989.

[49] GeneralFAQ < AFSLore < TWiki. Available from http://www.dementia.org/twiki/bin/view/AFSLore/GeneralFAQ Last visited 31.10.2007.

[50] Wachsman, A. april 2005. Part III – A Secure Distributed File System. *Linux Journal*. Issue 132.

[51] Többicke, R. 1994. Distributed file systems: Focus on Andrew File System/Distributed File Service (AFS/DFS). In *Proceedings of the thirteenth IEEE Symposium on Mass Storage Systems*.

[52] Anderson, T. Specification of FCrypt: Encryption for AFS Remote Procedure Calls. Available from http://surfvi.com/~ota/fcrypt-paper.txt Last visited 02.10.2007.

[53] Miller, S., Neuman, B., Schiller, J., & Saltzer, J. 1987. Section e.2.1: Kerberos authentication and authorization system. Technical report, MIT Project Athena.

[54] Trappe, W. & Washington, L. 2006. *Introduction to Cryptography with Coding Theory*. Pearson Education International, Upper Saddle River, NJ, USA, 2 edition, Pearson International Edition.

[55] Gollmann, D. 1999. *Computer security*. John Wiley & Sons, Inc., New York, NY, USA.

[56] TrueCrypt source code. Available from www.truecrypt.org/downloads.php.

[57] Barak, B. & Halevi, S. 2005. A model and architecture for psudo-random generation and applications to /dev/random. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, 203–212, New York, NY, USA. ACM Press.

[58] Viega, J. 2003. Practical random number generation in software. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, 129, Washington, DC, USA. IEEE Computer Society.

[59] SSH FileSystem. Available from http://fuse.sourceforge.net/sshfs.html Last visited 31.10.2007.

[60] Clercq, J. D. 2004. Kerberos advantages. Available from http://searchwindowssecurity.techtarget.com/originalContent/0,289142,sid45_gci 1009961,00.html Last visited 30.10.2007.

A Scripts

This appendix presents the script described in section 6.1, except random.sh and restorehead.sh, because of the simplicity of those scripts. In addition to this createC.sh is presented, to show the differences in the creation process in the version put in production at CERN.

Information specific for CERN, such as e-mail addresses, computer names and file paths have been excluded.

A.1 createvol.sh

```
#!/bin/sh
#####
#
# Script to simplify the creation of a new TrueCrypt volume.
#
# Run on the remote backup server.
#
# It is assumed that the new volume is not a partition.
# Folder where volume is to be stored (/volumes/) and mountpoint
# (/obhome/username) must be created before creation of volume.
#
# Name of the service account can be provided as a parameter to the script.
# If no name is provided, the user is prompted for it.
#
####
                       # Path to be used for backed up header and config file
afs_path="~/private/"
random=$afs_path"random.sh"      # Script to generate password
check=$afs_path"check.sh"        # Script used to check if volume is mounted.
                          # Path for backup up volume header on backup server
head_local="/user/truecrypt/"
volfolder="volumes"              # Folder for volumes
mountpath="/obhome/"             # Path for  mounting
volpath="/"$volfolder"/"         # Path for volumes

# Check parameters.
if test -z $1                    # Name of new volume is based on username
then
    echo "Service account: "
    read name
else
    if [ "$1" == "quick" ]       # In case name is not provided but quickformatting
    then quick=1
    else name="$1"
    fi
fi

if test -z $2                       # Check if quickformatting is to be used
then
   quick=0
else
    if [ "$2" == "quick" ]
    then quick=1
    else quick=0
    fi
fi

mountpoint=$mountpath$name
exist=$(ls / | grep $volfolder 2> /dev/null)
if [ "$exist" != "$volfolder" ]
then echo "Folder $vol_path does not exist. Please create it before trying again."
    exit;
fi

exist=$(ls $mountpath | grep $name 2> /dev/null)
if [ "$exist" != "$name" ]
then echo "Folder $mountpoint does not exist. Please create it before trying again."
    exit;
fi

password=$($random)
volume=$volpath$name                # Path for volume
info=$afs_path$name".txt"           # Path for configurationfile, with filename
printf "$volume" > $info            # Store volume name to use when mounting

# Creation of new volume - the new password is provided in the command
# The creator must also chose the size of the volume.
clear;
echo "A new TrueCrypt volume will now be created. Some interaction is necessary."
echo "Depending on the size of the volume, this process may take several hours.
To avoid this, use 'quick' option (may lead to decreased security). See the
documentation for more information."
echo "Press enter to continue, to abort press Ctrl+c."
read inn

if [ "$quick" == "1" ]
then
    truecrypt --quick --random-source /dev/random -p $password --filesystem
    fat --type normal --encryption AES --hash SHA-1 -c $volume;
else
    truecrypt --random-source /dev/random  -p $password --filesystem fat
    --type normal --encryption AES --hash SHA-1 -c $volume;
fi

echo "Backing up the volume header...."

afs=$afs_path$name".hea"         # Header file in AFS
local=$head_local$name".hea"     # Header file on backup server
exist=$(ls $local 2> /dev/null)  # Check if there is already a header file on
                                 # the backup server
if [ "$exist" == "$local" ]
then
    printf "\n\n\nA header file corresponding to this volume already exists.
            \nWould you like to overwrite, rename or cancel? (o/r/c): "
    read inn
    valid=0
    while [ $valid == 0 ]
    do
      case  $inn  in
        o)   truecrypt --backup-headers $local $volume;      # Backup header
             truecrypt --backup-headers $afs $volume;
             valid=1;;
        r)   date=$(date +%m-%d-%Y)
             rename_l=$local"_"$date
             mv $local $rename_l               # Rename file
             printf "\n\nThe old local header file is renamed to $rename_l.\n"

             exist=$(ls $afs 2> /dev/null)     # Check if there is also an
                                               # existing header file in AFS
             if [ "$exist" == "$afs" ]
             then
                 rename_a=$afs"_"$date
                 mv $afs $rename_a             # Rename file
                 printf "The old AFS header file is renamed to $rename_a.\n"
             fi

             truecrypt --backup-headers $local $volume;      # Backup header
             truecrypt --backup-headers $afs $volume;
             valid=1;;
        c)   printf "\n The volume header has not been backed up. This must
                    be done to ensure disaster recovery. Please see the
                    documentation for more information.  \n\n "
             printf "The password to the new volume is:\n\n "
             echo $password
             printf "\n\nPrint/write down the password and transport it to
                    the secure location according to the policies and
                    procedures, and keep a copy available for future use.
                    \nPress enter to continue.\n\n"
             read inn
             exit;;
        *)   echo "Invalid choice."
             echo "Would you like to overwrite, rename or cancel? (o/r/c):"
             read inn;;
      esac
    done
else
    truecrypt --backup-headers $local $volume;                # Backup header
    truecrypt --backup-headers $afs $volume;
fi

printf "\n\nThe password to the new volume is:\n\n "
echo $password
printf "\n\nPrint/write down the password and transport it to the secure
       location according to the policies and procedures, and keep a copy
       available for future use. Press enter when ready.\n"
read inn
printf "\n\nA keyfile to mount the volume will now be created. Please press
       'Enter' when prompted for existing keyfile and new password.\n"

keyfile=$afs_path$name".key"
truecrypt --random-source /dev/random --keyfile-create $keyfile;
truecrypt --random-source /dev/urandom -p $password --keyfile-add $keyfile -C $volume;

printf "$volume \n$mountpoint\n" > $info;        # Write info to file, to
                                                 # use with mount.sh++
```

A.2 createC.sh

```
#!/bin/sh
#####
#
# Script to simplify the creation of a new TrueCrypt volume
# in the CERN system design.
#
# This script must be run by root!!
#
# If the volume should be a separate partition, the hard disk must be
# partitioned before this process is started.
#
# Name of the new volume can be provided as a parameter to the script.
# If no volume name (name of partition) is provided, the user is
# prompted for it.
#
####

if [ "$USER" != "root" ];             # Check if the user is root. If not, quit.
then
    echo "You have to be root to create new volume."
    exit
fi

path="~/private/"
random=$path"random.sh"               # Script to generate password
check=$path"check.sh"                 # Script used to check if volume is mounted.
                               # Path for backup up volume header on backup server
head_local="/root/truecrypt/"

# Check parameters.
if test -z $1                     # Name of new volume / name of partition
then
    echo "Full path of new volume (name of partition): "
    read volume
else
    if [ "$1" == "quick" ]
    then quick=1
    else volume="$1"
    fi
fi

if test -z $2                      # Check if quickformatting is to be used
then
   quick=0
```

```
else
    if [ "$2" == "quick" ]
    then quick=1
    else quick=0
    fi
fi

password=$($random)

# Creation of new volume - the new password is provided in the command
# Hash and encryption algorithms must be chosen.
# If the volume is not a partition, the creator must also chose the size
# of the volume.

clear;
echo "A new TrueCrypt volume will now be created. Some interaction is necessary."
echo "Depending on the size of the volume, this process may take several
hours. To avoid this, use 'quick' option (may lead to decreased security).
See the documentation for more information.
Press enter to continue, to abort press Ctrl+c."

read inn

if [ "$quick" == "1" ]
then
    truecrypt --quick --random-source /dev/random -p $password --filesystem
    none --type normal -c $volume;
else
    truecrypt --random-source /dev/random  -p $password --filesystem none
    --type normal -c $volume;
fi

name=$(basename $volume)           # Strip volume name from path
info=$path$name".txt"          # Path for configuration file, with filename
printf "$volume" > $info           # Store volume name to use when mounting

echo "The volume will now be formatted to ext3. This will take some time.
Press enter to continue, to exit press x and enter".
read inn

if [ "$inn" == "x" ]
then
    printf "\n\nThe password to the new volume is:\n\n "
    echo $password
    printf "\n\nPrint/write down the password and transport it to the secure
           location, and keep a copy available for future use.
           Press enter to continue.\n\n"
    read inn
    printf "The volume is not formatted, the header file is not backed up and
           the information file is not created. Please see the instructions
           the documentation about manually creating a volume for information
           on how to complete the process.\n"
    exit
fi

export MKE2FS_SYNC=10;

truecrypt --device-number 99 -p $password $volume;
       # Set devicenumber 99 to ensure that no taken device numbers are used

control=$($check $name);
if [ "$control" == "1" ]
then
    /sbin/mkfs.ext3 /dev/mapper/truecrypt99;
    truecrypt -d;
else
    echo "Volume was not mapped. Formatting canceled. Press enter to continue
    with header backup. See the documentation for more information."
    read inn
fi

echo "Backing up the volume header...."
afs=$path$name".hea"                   # Header file in AFS
local=$head_local$name".hea"           # Header file on backup server

              # Check if there is already a header file on the backup server
exist=$(ls $local 2> /dev/null)

if [ "$exist" == "$local" ]
then
    printf "\n\n\n A header file corresponding to this volume already exists.
    \n Would you like to overwrite, rename or cancel? (o/r/c):"
    read inn
    valid=0

    while [ $valid == 0 ]
    do
      case  $inn  in
        o)   truecrypt --backup-headers $local $volume;      # Backup header
     truecrypt --backup-headers $afs $volume;
     valid=1;;
        r)   date=$(date +%m-%d-%Y)
```

```
             rename_l=$local"_"$date
             mv $local $rename_l                            # Rename file
             printf "\n\nThe old local header file is renamed to $rename_l.\n"

                      # Check if there is also an existing header file in AFS
             exist=$(ls $afs 2> /dev/null)
             if [ "$exist" == "$afs" ]
             then
                  rename_a=$afs"_"$date
                  mv $afs $rename_a                         # Rename file
                  printf "The old AFS header file is renamed to $rename_a.\n"
             fi

             truecrypt --backup-headers $local $volume;      # Backup header
             truecrypt --backup-headers $afs $volume;
             valid=1;;
        c)   printf "\n The volume header has not been backed up. This must
                    be done to ensure disaster recovery. Please see the
                    documentation for more information.  \n\n "
             printf "The password to the new volume is:\n\n "
             echo $password
             printf "\n\nPrint/write down the password and transport it to
                    the secure location, and keep a copy available for future
                    use. Press enter to continue.\n\n"
             read inn
             exit;;
        *)   echo "Invalid choice."
             echo "Would you like to overwrite, rename or cancel? (o/r/c):"
             read inn;;
      esac
    done
else
    truecrypt --backup-headers $local $volume;               # Backup header
    truecrypt --backup-headers $afs $volume;
fi

printf "\n\nThe password to the new volume is:\n\n "
echo $password
printf "\n\nPrint/write down the password and transport it to the secure
       location, and keep a copy available for future use. Press enter to
       continue.\n\n"
read inn


if [ "$control" == "1" ]
then
```

```
    echo "The volume will now be mounted. Please specify mountpoint:"
    read mountpoint
    truecrypt $volume -p $password $mountpoint
    printf "$volume \n$mountpoint\n" > $info
                                # Write info to file, to use with mountC.sh++
else
    printf "The volume was not formatted, and can not be mounted. Please
           format and mount the volume manually. See the documentation for
           more  information.\n Press enter to exit."
    read inn
fi
```

A.3 check.sh

```
#!/bin/sh
#####
# Script to control if a specific volume is mounted.
# Takes name of the volume as parameter.
# Information about the script is provided by the file
# ~/private/$volumename.txt
#
# Returns 1 if the volume is mounted, 0 if it is not.
#
# Run on the remote backup server.
#
# This script is the same for both system designs.
#
# Complete version: To be used by create.sh, mount.sh and umount.sh.
# CERN version: To be used by createC.sh and mountC.sh
#
#####

volumename=$1
info="~/private/"$volumename".txt"   # Path to configuration file.

# Read information about volume from file.

read volume < $info

# Check if volume is mounted

check=$(truecrypt -l 2>/dev/null | grep $volume | /usr/bin/awk '{print $2}');

if [ "$check" == "$volume" ]
    then
    echo "1"
else
    echo "0"
fi
```

A.4 mount.sh

```
#!/bin/sh
####
# Script to mount volume.
#
# Reads name of volume from the file $volume.txt
# Generates name of keyfile - $volume.key.
# Mounts the volume using the keyfile.
# Checks the capacity of the volume.
#
# Run on the remote backup server.
#
#####

path="~/private/"
check=$path"check.sh"
obmail=""

if test -z $1                # Name of volume
then
    echo "Service account: "
    read name
else
    name=$1
fi

volinfo=$path$name".txt"       # Path to configuration file

count=1
while read input; do           # Read configuration file
    if test $count -eq 1
    then
        volume=$input
    fi

    if test $count -eq 2
    then
        mount=$input
    fi

    count=$[$count + 1]
done < $volinfo

keyfile=$path$name".key"       # Path to keyfile

# Mounting volume:
```

```
truecrypt -u $volume -k $keyfile -p "" $mount;

# Check that mount was successful
result=$($check $name);

if [ "$result" == "1" ]
    then
    echo "Mount successful!"
else
    echo "Mount not successful. Please try again."
fi


# Check capacity
utilized=$(/bin/df -k $volume | /usr/bin/tail -1 | /usr/bin/awk '{print $5}');

thresh="75%"                    # Set threshold for usage

used=${utilized%%%}             # Trim %-sign from end of utilized and thresh.
limit=${thresh%%%}

if [ $used -gt $limit ]         # Check if the usage is larger than the threshold.
 then
    echo "ALERT!!! The volume is $utilized full!"
    echo "An e-mail is sent to the obadmins. Please consider creating a new
    volume, or future backups may be lost."
    echo ‘‘Volume $name is $utilized full. A new volume should be created for
    this user.’’ | mail -s ‘‘Volume $name is $utilized full’’
    $obmail;
fi
```

A.5 unmount.sh

```
#!/bin/sh
####
# Script to unmount volume.
#
# Reads name of volume from the file $volumename.txt
#
# Unmounts the volume.
#
# Run on the remote backup server.
#
#####

path="~/private/"
check=$path"check.sh"

if test -z $1                 # Name of volume
then
    echo "Name of volume: "
    read name
else
    name=$1
fi

volinfo=$path$name".txt"      # Path to configuration file

count=1
while read input; do          # Read configuration file
    if test $count -eq 1
    then
        volume=$input
    fi

    if test $count -eq 2
    then
        mount=$input
    fi
    count=$[$count + 1]
done < $volinfo

# Unmounting volume
truecrypt -d $mount;

# Check that unmount was successful
result=$($check $name);
```

```
if [ "$result" == "0" ]
    then
    echo "The volume has successfully been unmounted!"
else
    echo "The volume was not unmounted. Please try again manually. See
    documentation for further information"
fi
```

A.6 backup.sh

```
#!/bin/sh
#####
# Script to mount volume, run rsync command and unmount volume.
#
# Check if volume is mounted. If not, mount. If mounted, then procede with rsync.
# Takes username and directory to synchronize as parameter...
#
# Run on local (client) computer, connects to remote server via SSH
#
#####

path="~/private/"
check=$path"check.sh"
obmail=""
observer=""



if test -z $1
then
    echo "Service account: "
    read name
else
    name="$1"
fi

if test -z $2
then
    echo "Directory to synchronize: "
    read dir
else
    dir="$2"
fi

# Check if volume is mounted. If not, mount it.
echo "Checking if volume is mounted..."
read=$path"read.sh"                    # Script to read from file
volinfo=$path$name".txt"               # Name of configuration file
keyfile=$path$name".key";              # Name of keyfile

# Read information from configuration file on remote backup server
volume=$(ssh $name@$observer "$read $user $volinfo 1") #volumepath
mount=$(ssh $name@$observer "$read $user $volinfo 2")  #mount point

mounted=$(ssh $name@$observer "$check $name");
if [ "$mounted" == "1" ]
```

```
   then
       echo "Alert!! The volume was already mounted!!"
       echo "Synchronizing..."

       rsync -av $dir $name@$observer:$mount;

       # Unmount volume
       echo "Unmounting volume..."
       ssh $name@$observer "truecrypt -d $mount";

   else
       echo "Mounting volume..."
       ssh $name@$observer "truecrypt -u -k $keyfile -p '' $volume $mount";

       # Checking result of mounting, if not successful, nothing more happens.
       # If successful, rsync is run.
       mounted=$(ssh $name@$observer "$check $name");

       if [ "$mounted" == "1" ]
  then
    echo "Synchronizing..."
    rsync -av $dir $name@$observer:$mount;

    # Unmount volume
    echo "Unmounting volume..."
    ssh $name@$observer "truecrypt -d $mount";
       else
   echo "Mounting did not succeed. Please try again...."
       fi
fi

mounted=$(ssh $name@$observer "$check $name")
if [ "$mounted" == "1" ]
    then
    echo "The volume was not unmounted properly. The offsite backup
    administrators have been alerted."
    echo ``Volume $name not unmountd'' | mail -s ``The volume $name was
    not unmounted properly. Please do this manually.'' $obmail;
fi
```

A.7 changekey.sh

```
#!/bin/sh
#####
# Script to change keyfile of a volume.
#
# Takes username as parameter...
#
# Creates new keyfile and change keyfile related to volume.
#
# Run on remote server.
#
#####

path="~/private/"
volpath="/volumes/"

if test -z $1
then
    echo "Service account: "
    read name
else
    name="$1"
fi

volume=$volpath$name
keyfile=$path$name".key"
current=$keyfile"_old"

mv $keyfile $current

truecrypt --random-source /dev/random --keyfile-create $keyfile

truecrypt --random-source /dev/urandom --keyfile-add $keyfile
-k $current -p "" -C $volume

rm $current
```

B User guidelines for version in production at CERN

Procedure for mounting volume (e.g. after reboot)

- retrieve password
- login as “obadmin” on offsite backup server, using SSH.
- either
  - a) run script: mount.sh. Provide password when prompted. OR
  - b) mount manually using the following command:
    truecrypt <volumename> <mountpoint>
    where volumename is the name of the partition holding the encrypted data, and mountpoint is the desired access point for the information. Provide password when prompted.
    To check if the volume is correctly mounted, type
    truecrypt -l
    to list the mounted volume(s).

Procedure for changing password

- retrieve current password
- login as “obadmin” on offsite backup server, using SSH.
- Either:
  - run script: changepass.sh. OR
  - run command
    truecrypt ––random-source /dev/random -C [volumename]
    If not providing the volume name, enter it when prompted.
  - When prompted for keyfile path, press enter.
  - When prompted for current password, enter this.
  - When prompted for new keyfile path, press enter.
  - Enter new password when prompted. (Generate a random password of 20 characters using the script random.sh)
  - Confirm new password
  - Wait for confirmation

The new password are to be printed and kept securely by at least one person with privileges to log in as obadmin.

Procedure for unmounting volume

Either:
Unmount one specific volume:

- truecrypt -d /dev/mapper/truecryptX
  where X is a number for 0-9 and /dev/mapper/truecryptX is the device corresponding to the relevant volume. OR
- truecrypt -d <mountpoint>
  where mount point is the directory in which the data stored on the volume is accessible.)

OR
Unmount all volumes:

- truecrypt -d

NOTE: As long as there is only one volume on each backup server, truecrypt -d is sufficient.
Run the command
truecrypt -l
to check that the volume was successfully unmounted.

Procedure for creating a new volume

NOTE: This must be done as root, to be able to format the volume. If a partition is to be encrypted, the partition must be created before starting the TrueCrypt volume creation.

Either:

- Run script “createC.sh”

OR

- Create a password e.g. by using the script “random.sh”
- Run the command
  truecrypt ––random-source /dev/random ––filesystem none ––type normal -c <volumename>
  - If the new volume is not a partition, provide the size of the volume when prompted. (<2TB)
  - Chose hash- and encryption algorithm
  - Provide the password generated previously when prompted, then re-enter it to confirm.
  - When prompted for keyfile, press enter.
- Format the volume to ext3:
  - set the environment variable MKE2FS_SYNC to avoid the system freezing:
    export MKE2FS_SYNC=10
  - map volume:
    - truecrypt ––device-number 99 <volumename>

        - Provide password when prompted
    - format volume:
      /sbin/mkfs.ext3 /dev/mapper/truecrypt99
    - unmount volume (see Procedure for unmounting volume)
- Back up the volume header:
  Run the command:
  truecrypt ––backup-headers header_<name>.bac <volumename>
  where "name" is the name of the volume, but not the entire path (e.g. if the volume is /dev/mapper/volume, the name is "volume").
- Create a file “<volumename>.txt” with the contents
  <volumename>
  <mountpoint>
- Mount the volume (see Procedure for mounting volume)
- Print/write down the password and transport it to the secure location. Also keep a copy available (securely) for later use.

Possible issues when creating a new volume using “createC.sh”

Error message: "Volume was not mounted. Formatting canceled."
There was a problem when mapping the volume to /dev/mapper/truecrypt0. To format the volume, the device must be mapped. Map the volume manually, using the command
truecrypt –device-number 0 <volumename>
providing the password when prompted. To check that the volume was properly mapped, use the command truecrypt -l to list mapped/mounted volumes.
When the volume is mapped, format it using the command
/sbin/mkfs.ext3 /dev/mapper/truecrypt0
Unmap the volume (truecrypt -d) and mount it. (See Procedure for mounting volume.)

Using “quick” option when creating volume
When using the “quick” option, the volume is created without filling it with random data. This may be regarded as less secure because it will then be possible to retrieve such information as where data is written, how much data is written, etc. However, it does decrease the amount of time needed to create a new volume to that needed to enter the correct information and formatting the filesystem.

Restoring header

If the header of the volume is corrupted, it is possible to restore it from the backup.
Run the command
truecrypt ––restore-header <header file> <volumename>
Press enter when asked about volume type (normal).

C Correspondence with OpenAFS mailing list

```
Re: [OpenAFS] Encryption of traffic?

From: Russ Allbery (rra@stanford.edu)
Sent: Tuesday, June 26, 2007 6:39:14 PM
To: Katrine Svendsen (kat_svendsen@hotmail.com)
Cc: openafs-info@openafs.org


 Katrine Svendsen <kat_svendsen@hotmail.com> writes:

 > I’m trying to find some details about the traffic encryption between an
 > AFS client and server (fs setcrypt on/off), but this seems to be very
 > difficult.I would like to know about such things as keylength, mode of
 > operation, key generation/distribution etc. Does anybody have a good
 > source for this?

AFS uses an encryption method called fcrypt, which is a modified DES. Google
for fcrypt will return a lot of hits, although I don’t know if anyof them
have detailed analyses. This encryption method is fairly obsolete at this point.

> It also seems to me that not too much have happened in this field (when
> considering AFS) the last years. Am I right when I think that the
> network traffic-encryption in AFS is somewhat "ancient"? Why is there
> not more focus on this?

On the contrary, this is our top development priority apart from keeping
things generally working, and is the focus of both the rxk5 and rxgk work.
The difficulty is that replacing the encryption algorithm in AFSrequires
substantial protocol changes and ideally one wants to generalizethe encryption
layer and support all GSSAPI encryption types at the sametime, as well as
provide a framework for stronger authentication ingeneral. Both rxk5 and rxgk
have made substantial progress in the past year.


--
Russ Allbery (rra@stanford.edu) <http://www.eyrie.org/~eagle/>
```

```
Re: [OpenAFS] Encryption of traffic?

From: Marcus Watts (mdw@spam.ifs.umich.edu)
Sent: Tuesday, June 26, 2007 7:23:57 PM
To: Katrine Svendsen (kat_svendsen@hotmail.com)
Cc: openafs-info@openafs.org

 > Date: Tue, 26 Jun 2007 09:39:13 PDT
 > To: Katrine Svendsen <kat_svendsen@hotmail.com>
 > cc: <openafs-info@openafs.org>
 > From: Russ Allbery <rra@stanford.edu>
 > Subject: Re: [OpenAFS] Encryption of traffic
 >
 > Katrine Svendsen <kat_svendsen@hotmail.com> writes:
 >
 > > I'm trying to find some details about the traffic encryption between an
 > > AFS client and server (fs setcrypt on/off), but this seems to be very
 > > difficult.I would like to know about such things as keylength, mode of
 > > operation, key generation/distribution etc. Does anybody have a good
 > > source for this?
 >
 > AFS uses an encryption method called fcrypt, which is a modified DES.
 > Google for fcrypt will return a lot of hits, although I don't know if any
 > of them have detailed analyses. This encryption method is fairly obsolete
 > at this point.
 >
 > > It also seems to me that not too much have happened in this field (when
 > > considering AFS) the last years. Am I right when I think that the
 > > network traffic-encryption in AFS is somewhat ``ancient`''? Why is there
 > > not more focus on this?
 >
 > On the contrary, this is our top development priority apart from keeping
 > things generally working, and is the focus of both the rxk5 and rxgk
 > work. The difficulty is that replacing the encryption algorithm in AFS
 > requires substantial protocol changes and ideally one wants to generalize
 > the encryption layer and support all GSSAPI encryption types at the same
 > time, as well as provide a framework for stronger authentication in
 > general.
 >
 > Both rxk5 and rxgk have made substantial progress in the past year.
 >
 > --
 > Russ Allbery (rra@stanford.edu) <http://www.eyrie.org/~eagle/>>

 > OpenAFS-info mailing list
 > OpenAFS-info@openafs.org
 > https://lists.openafs.org/mailman/listinfo/openafs-info

Fcrypt is described here: http://surfvi.com/~ota/fcrypt-paper.txt
it was probably obselete when deployed; its only real advantage is speed.
It's used in cbc mode, and there's some sort of not terribly convincing
checksum algorithm happening as well. AFS also supports ``integrity
without privacy'' (rxkad_auth), which does not encrypt the payload but can
detect changes, and ``rxkad_clear'', which is what you get for fileserver
traffic if you say ``fs setcrypt off''.

As designed and originally deployed, AFS used kerberos 4 - DES + pcbc.
There are lots of references out there for kerberos 4 & pcbc. AFS uses
the des session key straight with fcrypt; there's no further key
generation happening.

Current distributions of AFS also support kerberos 5 - but only with DES.
That is the service key & the session key must both be DES. The rest of
rxkad works exactly as before.

A patch for rxk5 is here:
/afs/umich.edu/group/itd/build/mdw/openafs/patches/afs-rxk5-r1518-m50.patch.bz2
it adds in support for kerberos 5 with mit or heimdal, and supports current
kerberos 5 encryption types. The token interface between the kernel & userland
will change slightly shortly (at the request of the rxgk folks), and there
is still some other cleanup to happen. rxk5 generates a different key for each
packet sent & received, and uses straight kerberos 5 encryption protocols to
either encrypt (rxk5_crypt) or do integrity but no privacy (rxk5_auth).
rxk5 does not by default support ``no integrity'' (rxk5_clear) and the small
bits of the fileserver & cache manager that known about this select rxk5_auth
for rxk5 in place of rxkad_clear for rxkad.


Help, I'm being dragged off to a meeting! Hope this helps...


-Marcus Watts
```