TransCrypt: an Enterprise Encrypting File System over NFS

Abhay Khoje, Salih K A and Rajat Moona

***Abstract*— Many organizations have great deal of confidential information which is stored on computers. Such information is desired to be kept securely yet giving a convenience of accessibility from any part of the world. For data security, one can use an encrypting file system such as eCryptfs [1], dmCrypt [2], File Vault [3]. However these encrypting file systems do not address the problem of accessing files over network from public computers. In this case the public host, the actual FileServer host and the network between them are vulnerable to many attacks. This paper discusses the major problems and proposes a solution for the same using TransCrypt [4] encrypting file system. It also describes how the proposed solution can be implemented on a Linux-based environment.**

***Index Terms*— FileServer (FS), WorkStation (WS), Encrypted File System (EFS), TransCrypt, SmartCard (SC), Network File System (NFS).**

I. Introduction

Many organizations and institutions use centralized storage devices. As a consequence of this, securing confidential data against thefts which eventually impose risks of losing important personal and organizational data, is of utmost importance. Such attacks include the attacker getting access to files on shared storage, modifying the contents of those files etc. Using strong Cryptographic techniques these attacks can be avoided by keeping confidential data in an encrypted manner. Therefore, there is a need for a storage solution which uses strong cryptographic methods to protect data.

Several solutions have been devised to address this problem. This include File Vault [3], eCryptfs [1], dm-crypt [2], Microsoft EFS [5] etc. These encrypting file systems provide security by encrypting and decrypting user data thereby addressing the problem of data security in many different ways, such as per-file encryption, flexible key sharing and inclusion of superuser in the trust model. Another encrypting file system *TransCrypt [4]* was designed to provide a very strong solution to the problem of securing data in a user transparent manner by our group.

TransCrypt is an enterprise-class, kernel-space encrypting file system for the Linux operating system, which incorporates an advanced key management scheme to provide a high grade of security, while remaining transparent and easily usable. It provides per-user access control, per-file encryption and per-volume managerical control. However it is not designed to work over network and hence does not address the security issues when the data is accessed over network from public computers.

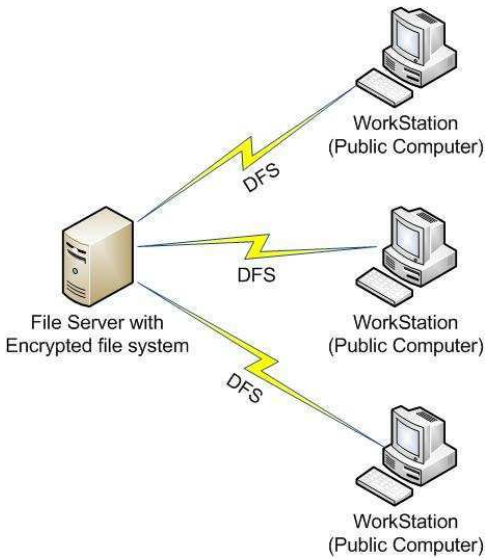

Fig. 1. Model of file access in Public Networks

In this paper, we discuss the problems that arise when secure data is accessed from public computers over network and propose a solution for the same using *TransCrypt* encrypting file system. We assume the model shown in Fig 1 to develop our solution. In this scenario, WS, FS and the network channel between them are vulnerable to many attacks like user masquerading attack, replay attack, man-in-the-middle attack and client spoofing. We make use of known cryptographic methods like mutual authentication, process credentials and session establishment to counter various attacks.

II. Previous Work

TransCrypt [4] is a secure, usable, transparent, efficient enterprise-class encrypting file system for Linux. It has stronger trust model and hence stronger security compared to other existing encrypting file systems. In this system only the kernel remains under the trust domain of the user. TransCrypt uses per-file cryptographic keys for flexible file sharing. It includes a set of simple userspace daemons and support utilities for administration. It reuses pre-established enterprise security frameworks such as Public Key Infrastructure for user

Manuscript submitted on March 24, 2009. This work was supported by Prabhu Goel Research Centre for Computer and Internet Security, Indian Institute of Technology Kanpur.
Abhay Khoje is a Master's student in the Department of Computer Science and Engineering, IIT Kanpur, India. (email: akhoje@cse.iitk.ac.in).
Salih K A is a Master's student in the Department of Computer Science and Engineering, IIT Kanpur, India. (email: kasali@cse.iitk.ac.in).
Rajat Moona is a Professor in the Department of Computer Science and Engineering, IIT Kanpur, India. (email: moona@cse.iitk.ac.in).

access control and provides support for data recovery, backups etc.

In the initial version of TransCrypt [4], the implementation was tightly coupled with the ext3 file system. TransCrypt modified on-disk structures to store additional meta-data required for its functionality. These modifications also included changes to the userspace e2fsprogs package which contains libraries and tools to work with ext3 volumes. A tight integration with ext3 file system made TransCrypt incompatible with other advanced file systems. The newer version of transcrypt was re-architectured [6] to employ a layered architecture, with its layers being file system independent. It used the extended attributes support provided by the file systems for metadata storage. With this mechanism, a great deal of flexibility and code maintainability is achieved.

In addition, the earlier implementation of TransCrypt [4] also modified the file I/O functionality of the kernel. Hence it had several performance and maintenance related limitations. Additionally, TransCrypt implementation used page cache for its operations. The direct I/O operations where page cache is bypassed, could not be supported by TransCrypt. For perfomance reasons, direct I/O operations are extensively used by applications such as database management systems etc. The new version of TransCrypt [7] uses a layer of cryptography to achieve modularity and direct I/O support [8]. This support is based on the device-mapper infrastructure [2] of the Linux kernel.

III. Issues

The default authentication mechanism of NFS [9] is *UID based*, when user accesses files over network. On the other hand, TransCrypt authenticates the user by sending a challenge, the response to which is computed by the user's private key store (PKS). The TransCrypt authentication mechanism is much more stronger than the UID based authentication as it uses *asymmetric key cryptography*. However TransCrypt is not designed to work over the network. Current implementation of TransCrypt looks for user's PKS only on the FS and not on WS where the user has logged in and hence sends the user authentication messages to FS instead of WS. The TransCrypt design is also vulnerable to many other attacks while accessing files over network from a workstation (WS). This section discusses the major attacks in detail.

A. Masquerading attack

A masquerading attack takes place when an entity poses itself as another entity. This is the attack in which a privileged user can acquire the identity of another user (e.g. using 'su -') and can access the files of a genuine TransCrypt user. The NFS suffers from this problem due to its UID based authentication.

This attack is illustrated in Fig 2. Let us assume that user *usr1* is logged in on machine *WS1* with its smart card on machine *WS1*. At the same time user *usr2* can get maliciously access to files of *user usr1* if it has superuser credentials on a machine *WS2*. For this, *usr2* logs in to machine *WS2* and acquires the identity of *usr1* (using 'su - usr1'). In order to get access to files of usr1, usr2 would send an access request to FS host as usr1. FS host uses the TransCrypt access control mechanism to get access to the operations with private key of usr1 (earlier registered at WS1) and thus provides access to usr2.

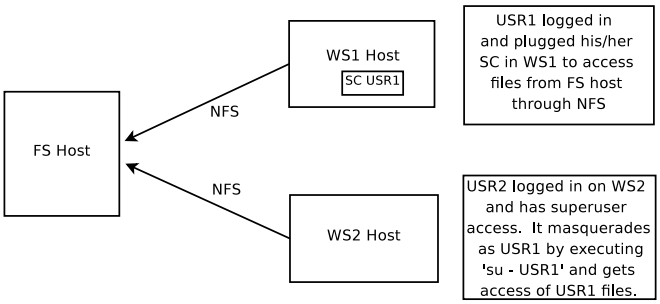

Fig. 2. Masquerading attack over Network

B. Man-in-the-middle attack

In this attack, an attacker actively monitors the traffic, captures and controls the communication transparently between two entities. For example, an attacker can re-route a data exchange. When computers are communicating at low levels of the network layer, the computers might not be able to determine with whom they are exchanging data. Attacker makes independent connections with two entities and relays messages between them. The attacker makes them to believe that they are talking directly to each other over a private connection when in fact the entire conversation is seen and controlled by the attacker.

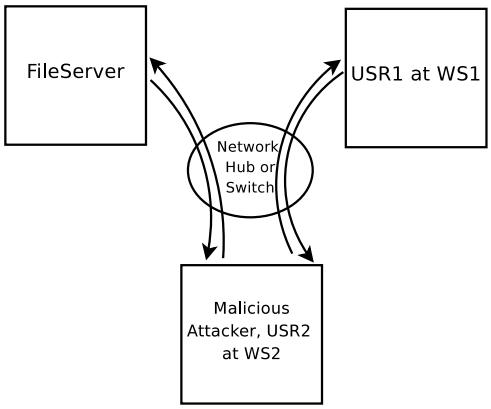

Fig. 3. Man In the Middle attack

Consider Fig 3, where user *usr1* on machine *WS1* wants to access its files from *FS* over the network. Meanwhile, user *usr2* on machine *WS2* may eavesdrop on the conversation. To get started, *usr1* sends the file identifier and his user ID to *FS* over the network. The user *usr2* is able to intercept it, thereby initiating a man-in-the-middle attack. *usr2* sends a forged message to *FS* claiming to originate from *usr1* at WS2. FS, believing this message to originate from *usr1* at WS2, sends the response to WS2. Now *usr2* depending on his desire, can send altered data to *WS1*. When *usr1* receives the message, it believes the response to come from the genuine

FS. In this case a malicious user got access to a genuine user's file and is even able to alter his data.

C. Replay Attack

Replay attack involves the re-use of captured data at a later time than originally intended, in order to repeat some action of benefit to the attacker. In our scenario, the attacker can capture the packets containing file handle information (of a genuine user) and can replay it later to get access to those files.

IV. Our approach

Current implementation of TransCrypt looks for user's PKS only on the FS and not on WS where user is logged in. This can be solved by registering user's PKS with the FS kernel. FS kernel will store this location and during subsequent file operations, authentication messages will be sent to the user's registered PKS location.

This section describes an approach to solve the attacks when users access files over network from a workstation using NFS.

A. User masquerading attack

NFS is prone to user masquerading attack. Hence processes of the attacker session can have the UID same as that of the genuine user (usr). UID based authentication is not sufficient to protect TransCrypt volume from this attack. FS kernel must have mechanisms to differentiate between the genuine session and a masqueraded session in order to mitigate this attack. To differentiate between these two sessions, we need to establish some unique credentials between the user login process and the FS kernel. During any file access, these credentials will also be sent to the FS. FS kernel will verify these credentials and give access to only those operations coming from a genuine user. Since the masqueraded sessions won't have the correct credentials, it won't get access to the files.

B. Man-in-the-middle attack

This attack is solved by establishing a session key between the WS and the FS. All further communication between the hosts will go encrypted with this symmetric key along with message integrity code. The attacker cannot have this session key and hence cannot interpret the messages or modify them without being noticed.

C. Replay Attack

This attack is solved by sending the file operation responses to the genuine user, encrypted with a freshly generated session key. Only the genuine user will have the correct credential and the session key. These keys (session key and credential) are established at log in time to maintain the freshness so that replay attacks are avoided.

V. Protocols

This section describes the *Trust Model* which we assumed in developing the protocol, *the Negotiation protocol* which establishes the common cryptographic parameters (between FS and WS) and *the secure protocol* which enables transcrypt work over network. Secure Protocol registers the user PKS, perfoms mutual authentication and establishes session key between FS and WS kernel.

A. Trust Model

There are two main entities in our model, namely Work Station and FileServer. Trust model assumed for these entities are described below.

1) WorkStation
   - Super user on WS is partially trusted not to substitute the kernel image with a malicious version over a system reboot.
   - WS login program is trusted to assign randomly generated credentials provided by the FS to correct login session only and not to any malicious user's login session.
   - WS kernel is trusted, not to assign credentials to processes created by malicious user and to send the correct credentials when user is accessing transcrypt volumes.
2) FileServer
   - FS kernel is trusted not to leak file encryption keys and the file system keys used in TransCrypt.
   - Super user on FS is partially trusted not to substitute the kernel image with a malicious version over a system reboot.

B. Negotiation Protocol

Our solution is based on strong cryptographic methods to support mutual authentication and establishment of session keys between FS and user's PKS. It makes TransCrypt robust enough to work over network. For this protocol to work, the PKS and FS should agree upon the PKI algorithms and the lengths of cryptographic challenges prior to the start of communication.
PKS can be any secure device such as SmartCard, Mobile Phone, USB tokens etc., which supports public key infrastructure (PKI). We assume that SmartCard is the PKS in our environment, since it is one of the most secure tool to keep secret keys. Fig 4 explains the negotiation protocol in detail.

When a SmartCard (SC) connects to the reader, the WS sends a *HELO* message with a list of supported parameters such as challenge lengths and set of algorithm identifiers for various cryptographic algorithms to the FS. Algorithm identifier consists of algorithm name and other parameters like key length, cipher specification etc. When FS gets the *HELO* message, it selects the most suitable algorithm and parameters from among those that it received from the client and sends a *WELCOME* message (which contains the chosen parameters) to the SC.

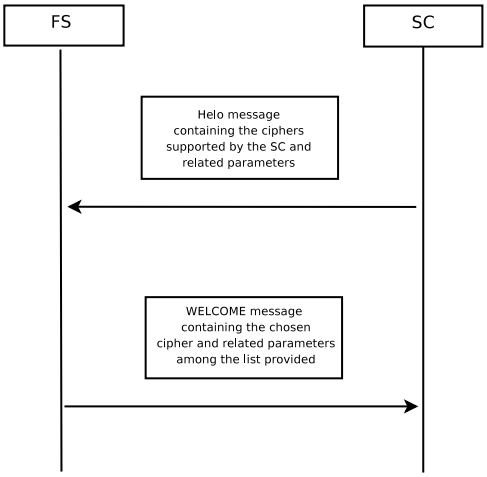

Fig. 4. Negotiation Protocol

C. Secure Protocol

The secure protocol enables transcrypt to work over NFS. It consists of mainly two steps.

1) PKS Registration: Location of the user's PKS is conveyed to Transcrypt for sending the authentication messages when user accesses the transcrypt volumes. This is done by registering user's PKS location to the FS kernel. FS kernel stores this location (known as SCURI for Smart Card Uniform Resource Identifier) and sends authentication messages for file access to the corresponding user's PKS.

The FS host needs the identification of the user logged on the WS before giving access to the files. On NFS, this identification is provided by a common UID. However this is not possible in our case since the WS host is not under the same administrative control. For the purpose of user's identification we use the certificate of the user that is provided by the user's SC. The certificate is first verified and the user's name is extracted from the certificate by the FS host. The login program on the WS host facilitates this process by reading the certificate from the SC and sending the user's name along with the UID at WS host and a uniform resource identifier to locate the user's SC (SCURI) to the FS. The FS locates the certificate from its repository by matching the certificate name and verifies the validity of the certificate.

2) Authenticate and establish a session between WS kernel and FS kernel: To the counter various attacks mentioned earlier, the FS and WS need to authenticate each other and establish session keys for encryption and message integrity before any file operations commence. The session keys are generated using equal contributions of key material from two sides.

The entire process is shown in Fig 5. In the figure E^P_U, E^P_{FS} ,E^P_{WS} denotes Public Key encryption using the public key of user U, file server FS and work station WS respectively.

1) FS kernel sends a random challenge (r_fs) and its contribution of the session key (k_fs) to the WS login process encrypted with the public key of the user.
2) WS login process forwards the message to the user's SC.
3) The SC decrypts the message using the private key of the user. It sends back the response to the FS challenge (r_fs), a random challenge (r_sc) and its contribution of the session key (k_sc) to the WS login process encrypted with the public key of the FS.
4) WS login process forwards the message to the FS.
5) FS kernel decrypts the message using its private key. It verifies the challenge response and if it is succeessful, user is authenticated to FS, else the protocol is aborted. After authentication, the FS kernel generates a random and unique credential (SID) for this particular user session. It then sends back the credential (SID) and the SC challenge response (r_sc) to WS login process encrypted with the public key of the user. FS kernel now computes the session key (SK_fs_ws) using the key materials (k_fs and k_sc) and maintains the state information, (UID, SCURI, SID, SK_fs_ws) in a table for future use.
6) WS login process forwards the same packet to the SC.
7) SC having the private key of the user decrypts the message. It verifies the challenge response (r_sc) and If it is successful, FS is authenticated to the user, else the protocol is aborted. After authentication, the SC computes the session key (SK_fs_ws) using the key materials (k_fs and k_sc). The WS has to be authenticated to the user before passing the session key and the credential to the WS kernel. The WS certificate is manually verified by the user and is passed to the SC.
8) SC sends a random challenge (r) to the WS encrypted with the public key of the WS host.
9) WS kernel decrypts the message and sends back the response (r) to the SC.
10) At this point, WS is authenticated to the SC. SC now passes the credential (SID) and the session key (SK_fs_ws) to the WS login process encrypted with the public key of the WS host.

WS login process now passes these blinded SID and SK_fs_ws to the WS kernel. WS kernel extracts the keys (using its private key) and assigns the credential to the user's login process so that all of its children inherit the same credentials.

VI. NFS OPERATIONS

During any file operations such as open, read, write by the user, the NFS client sends the credential (SID) along with the other standard parameters to the FS encrypted with the session key (SK_fs_ws). FS decrypts the message using the session key and verifies the credential. If the verification fails, FS does not process the request further. FS also checks whether the request came from the WS to which the SID is associated and aborts the request if not. Fig 6 shows these steps in detail. E^S denotes Symmetric Key encryption in the figure. In case of the read and write operations, the data to be read or to be written can even be encrypted using IPsec or the session key established (if needed).

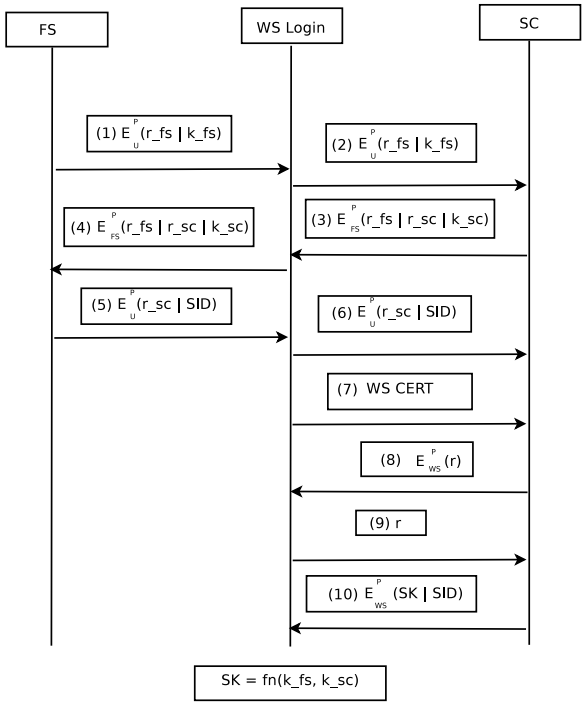

Fig. 5. Authentication Protocol

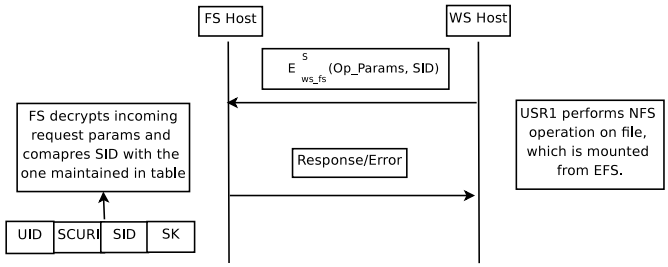

Fig. 6. NFS file operations

VII. Implementation

Our implementation of the protocol is a scalable one. It involves changes in the NFS server as well as in the NFS client.

A. NFS Server Changes

NFS server side changes include changes in certain components of TransCrypt and changes in the NFS file system.

1) TransCrypt File System Changes: The user space as well as kernel space components of TransCrypt are modified to support credentials based executions. In the userspace we modify the TransCrypt daemon to support initial authentication protocol. In the kernel space we modify the TransCrypt to keep the state information corresponding to every authenticated user.

Each of these changes are explained in brief below.

- *TransCrypt Daemon*. All communication between FS kernel and other entities (login process, SC) are routed through TransCrypt daemon. The modified TransCrypt kernel must keep a record of the PKS location of the user. Therefore it must receive the PKS registration messages as part of the secure protocol. To receive the user's PKS registration packets, TCP server functionality is added to the current version of TransCrypt Daemon.
- *Netlink Communication Manager*. In the current TransCrypt version, the communication manager subsystem supports only kernel initiated packets. It is not capable of handling user initiated requests like those in the authentication protocol. The communication manager is modified so that the messages related to different protocols are handled appropriately.
- *User Authentication Subsystem*. This subsystem generates the messages that are part of the initial authentication protocol and then communicates them to the NFS Client through the communication manager. It also stores the credentials, user details and the session keys established as part of the protocol in a shared data structure.

2) NFS file system changes: It includes changes in the NFS file system. All NFS requests are handled by a function "permission" to check for the genuineness of the request. We modified this function, to verify whether the NFS client requests originate from a genuine user session. If not, the request is rejected.

B. NFS Client changes

The NFS Client implementation needs certain changes to support the modified protocol. These changes are done in user space as well as kernel space components of the NFS client. The changes in the user space components of the NFS client include the mechanisms to support the login protocol and the support to contact SC through the smart card daemon. The changes in the kernel space components include the mechanism to assign the credentials to the processes as explained earlier. These changes are described below.

- *Login Patch*. The changes in the login process must incorporate the mutual authentication between NFS sever and client. This is done using the Pluggable Authentication Module (PAM) [10]. This module provides two methods to perform tasks associated with session set-up and to perform tasks associated with session tear-down. In our implementation session set-up method is used for mutual authentication and session key establishment (between User's SmartCard and FS kernel). It also passes the per-user session credentials obtained from the SC to the WS kernel through a virtual I/O driver. During the session tear-down, the FS kernel is informed about the session termination so that it may remove the corresponding state information.
- *SmartCard Daemon*. It has to take care of the full duplex communication between NFS Client, login process and user's SC. It integrates PKCS #11 APIs [11] to communicate to the SC and socket support to communicate with other entities (login process, transcrypt-daemon).
- *Device Layer in kernel*. The login process passes the credentials to the kernel for assignment to the processes as explained earlier. Such changes are done through an installable device driver. The login process uses an ioctl [12] call to perform this functionality.

- *NFS file system changes*. During any file operation from the NFS Client, WS kernel will pass the credentials corresponding to the user to the NFS server. This along with other parameters is relayed to the FS through the secure channel established earlier between NFS server and client. These modifications are done by patching the NFS client code in the Linux kernel.

VIII. Conclusion

In this paper we have discussed the security vulnerabilities, when a user tries to access his files over an untrusted network. We also proposed a secure and scalable protocol to solve the same. The strong trust model in our implementation where even administrators are not trusted distinguishes our solution from other existing solutions. We have a scalable implementation of the solution in Linux environment using the open source encrypted file system, TransCrypt [4]. The modularized implementation helps an integration of the solution to any environment. Our implementation is open source and can be obtained through the home page of TransCrypt [7].

IX. Acknowledgment

We would like to thank Prof. Arnab Bhattacharya and Prof. Piyush Kurur, Deparment of Computer Science and Engineering, IIT Kanpur for their help and suggestions through out our work. We also thank Satyam Sharma, Doctoral student, Department of Computer Science and Engineering, IIT Kanpur for all his guidance.

References

[1] M. A. Halcrow, "eCryptfs: An Enterprise-class Encrypted Filesystem for Linux," in *Proceedings of the Linux Symposium*, Ottawa, Canada, Jul. 2005, pp. 201–218.
[2] "dm-crypt: a device-mapper crypto target for linux," Available, http://www.saout.de/misc/dm-crypt/.
[3] "Apple Mac OS X FileVault," Available, http://www.apple.com/macosx/features/filevault/.
[4] S. Sharma, R. Moona, and D. Sanghi, "TransCrypt: A Secure and Transparent Encrypting File System for Enterprises," in *8th International Symposium on System and Information Security*, 2006.
[5] "How Encrypting File System Works," Available, http://technet2.microsoft.com/WindowsServer/en/Library/997fdd99-73ec-4041-9cf4-1370739a59201033.mspx.
[6] A. Raghavan, "File System Independent Metadata Organization for Transcrypt." Master's thesis, Indian Institute of Technology Kanpur, India, Available, www.cse.iitk.ac.in/ moona/students/Y6111006.pdf.
[7] "TransCrypt Filesystem Homepage," Available, http://www2.cse.iitk.ac.in/transcrypt/.
[8] S. Vellal, "A Device Mapper based Encryption Layer for Transcrypt." Master's thesis, Indian Institute of Technology Kanpur, India, Available, www.cse.iitk.ac.in/ moona/students/Y6111039.pdf.
[9] R. Sandberg, D. Goddberg, S. Kleiman, D. Watch, and B. Lyon, "Design and Implementation of the Sun Network File System," in *Usenix Conference Proceedings*, June 1985.
[10] V. Samar and R. Schemers, "Unified Login with Pluggable Authentication Modules," in *Conference on Computer and Communications Security*, March 1996.
[11] "PKCS 11: Cryptographic Token Interface Standard." Available, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/.
[12] "ioctl - control device," Manual Page, `ioctl(2)`.