Key Management in an Encrypting File System

Matt Blaze
AT&T Bell Laboratories

Abstract

As distributed computing systems grow in size, complexity and variety of application, the problem of protecting sensitive data from unauthorized disclosure and tampering becomes increasingly important. Cryptographic techniques can play an important role in protecting communication links and file data, since access to data can be limited to those who hold the proper key. In the case of file data, however, the routine use of encryption facilities often places the organizational requirements of information security in opposition to those of information management. Since strong encryption implies that only the holders of the cryptographic key have access to the cleartext data, an organization may be denied the use of its own critical business records if the key used to encrypt these records becomes unavailable (e.g., through the accidental death of the key holder).

This paper describes a system, based on cryptographic "smartcards," for the temporary "escrow" of file encryption keys for critical files in a cryptographic file system. Unlike conventional escrow schemes, this system is bilaterally auditable, in that the holder of an escrowed key can verify that, in fact, he or she holds the key to a particular directory and the owner of the key can verify, when the escrow period is ended, that the escrow agent has neither used the key nor can use it in the future. We describe a new algorithm, based on the DES cipher, for the on-line encryption of file data in a secure and efficient manner that is suitable for use in a smartcard.

1. Introduction

Modern distributed computing systems, for all their virtues, make it difficult to limit reliably access to sensitive data. Networks often unselectively broadcast data to far-reaching and unpredictable places, remote login facilities create new opportunities for trespassers and distributed file systems often assume that all machines to which they provide service are trustworthy and reliable. To reduce these risks, cryptographic techniques make it possible to limit data access while still taking advantage of untrustworthy networks and services. Modern workstations can encrypt in software at close to network speeds [4][5]. Data encryption attempts to ensure that only those who possess the correct decryption key can obtain the cleartext data.

Most commercial applications of encryption techniques protect communication links (and related services such as electronic mail). When communication endpoints are under the control of a single entity, or trust a common authority, the management of cryptographic keys is a conceptually straightforward matter. Keys can be assigned and changed as often as desired, the main problem being to ensure that both sender and receiver agree as to the current keys and that keys are discarded when no longer in use. Should sender and receiver get "out of sync" with the keys, the problem becomes immediately apparent because communication fails. Ensuring access by third parties in the event that keys are lost or unavailable is rarely an issue.* Public key techniques [3][10] make communication key management easier, allowing two parties to establish a secure channel without prior arrangement.

*The law enforcement community argues that it may be an exception; widespread use of encryption techniques may impede police wiretap investigations [2]. The ethical, legal, social and technical implications of law enforcement access to cryptographic communication are presently the subjects of intense public debate in the United States and are (fortunately) outside the scope of this paper.

Cryptography can also be used to protect file data, although there are relatively few tools for this purpose in widespread use. Most file encryption takes place at the application level, with tools such as the Unix **crypt** command or with special encrypting applications (e.g., "**vi -x**"). File encryption can also take place at a lower level, as a basic service of the file system [1][9][13].

Regardless of where encryption takes place, key management for encrypted files is a fundamentally different problem from that in cryptographic communication. In a secure communication system, keys must be distributed and synchronized *geographically.* Keys often serve the dual purpose of authenticating identity as well as protecting against eavesdroppers. The architecture for distributing communication keys is closely tied to the trust relationships within the system, and practical key distribution protocols (such as those employed by the Kerberos system[12]) must be carefully engineered to balance reliability, security and performance.

In a file system, on the other hand, there is usually little need to distribute keys geographically; most protected files are encrypted and decrypted at the same locations (and by the same users). Authentication of identity is a less serious issue, with access implicitly controlled through knowledge of the key itself, although cryptographic techniques can also be used to detect unauthorized tampering with file data. File systems still present a significant, if differently formulated, key management problem, however, in that keys can be said to be distributed *temporally.* The corresponding keys must be available at both encryption and decryption time. File encryption keys have much longer lifetimes than their communication counterparts. If a key is lost or unavailable, the files encrypted with it are rendered useless. This condition may not be detected until it is too late. The key distribution center and public key cryptographic protocols developed for geographically distributed communication systems do not have direct analogues that can be readily applied to temporal file key management.

Arguably, it is because of difficulties associated with key management that sensitive files are rarely encrypted in practice even when encryption tools are available. This is especially true in critical business environments where ensuring the availability of data to authorized users is at least as important as ensuring its unavailability to everyone else. Sometimes, files are protected with weak ciphers, such that the encrypted data can be recovered with the application of sufficient computing resources. A toolkit ("Crypt Breaker's Workbench") is available in Internet archives for the purpose of decrypting files enciphered with the Unix **crypt** program. Needless to say, since these tools are also available to the adversary, encryption with weak ciphers is of questionable value in the first place.

In the context of organizational information systems, cryptographic file protection presents several problems not addressed by traditional (communication-oriented) key management schemes. These problems are not only technical (e.g., providing mechanisms for ensuring that keys are available when and where authorized) but also managerial and social (balancing secrecy and privacy against emergency access requirements). Carefully controlled key management services with explicit, auditable trust relationships that are integrated into the underlying file system security architecture can help reconcile these often conflicting goals.

2. Key Escrow

Hence the problem: strong file encryption is often necessary to protect privacy while availability requirements sometimes dictate the need for a "back door" for emergency access. We use as our model the common problem of ensuring continued access to critical business files even after the only employees who know the keys to those files leave the organization. One approach adapts the procedures used for controlling physical locks and keys to file encryption keys and provides a central key distribution ("locksmith") service. Any time a user requires an encryption key, it is generated by a central service, which also keeps a copy for emergency access.

In practice, however, the central locksmith model adapts poorly to large-scale file encryption key management. The central service must be unconditionally trusted by all who obtain keys from it. No further controls preclude or audit access by those with access to the key database. (Note that this is not the case with locksmiths who manage physical keys — use of a key requires access to the lock, which may itself be controlled by independent security mechanisms and which can be changed if the locksmith's office is compromised. In the case of file keys, on the other hand, once a copy of the key database has leaked, all files with keys in the database must be considered compromised forever.) Furthermore, a central service can quickly become a service bottleneck or worse, a single point of failure or attack. The key service is an "online" part of the key creation process and users cannot create new keys if the service is unavailable. Finally, the problem of securing communication between the user and the key center introduces all the problems of communication key management in addition to the existing problem of file key management.

An alternative approach reverses the relationship and provides a controlled mechanism for users to deposit copies of their keys for emergency use as needed. The keys for crucial files could thereby be "escrowed" with a trusted caretaker who would reveal them only when certain conditions are met, such as when encrypted business data are required after the death of the legitimate key holder. Conceptually, keys might be delivered within sealed "envelopes." When a set of files is no longer critical, the envelope containing its keys could be returned to its originator, who could verify the integrity of the seal and destroy the keys, preventing future access to outdated, but still private, data. The "escrow-deposit" approach has the benefit of allowing the key holder to generate keys in the usual manner, without direct "online" interaction with a third party. There is no central service bottleneck, since the escrow agent is not directly involved in the creation of new keys. Envelopes containing escrowed keys can be delivered to the escrow agent at any time and any inability to deliver the keys to the agent need not preclude their use by the key holder.

Unfortunately, this is difficult to do in practice. The simplest procedure has the key holder write down the key, place it in a sealed envelope, and leave it with a trusted caretaker. This is vulnerable to mistakes, however, since there is no inherent mechanism to ensure that the escrowed key is the same as the real one. The security of the scheme also depends entirely on the honesty of the caretaker and the tamper-resistance of the envelope. An electronic analogue to the sealed envelope can be implemented by encrypting the key with a "caretaker" key, perhaps using public key techniques. If this is done automatically as part of key generation, the problems associated with transcription mistakes are avoided, but the scheme still depends entirely on the caretaker's honesty (and even more so without the sealed envelope). If no single caretaker can be trusted, the key could be multiply encrypted with more than one caretaker's key, split among several escrow agents (in the manner of the US Escrowed Encryption Standard) or encrypted using a group-oriented public key protocol.

Both the manual and encrypted key escrow schemes suffer from a fundamental problem, however. After an escrow agent "opens" the key and learns its value, no further controls on its use are possible. Anyone who learns the keys can use them at any time in the future without detection. Electronic escrow is particularly hard to revoke or audit, since it is difficult to ensure that all copies of the keys have been destroyed when the escrow period ends even if the keys have never actually been used (consider backups or illicit copies of the escrow data).

Under these schemes, key escrow is an "all or nothing" proposition, with no mechanism to guarantee, in any formal sense, that the caretaker is doing his or her job honestly. It is not obvious how to implement key escrow schemes that offer stronger protection against abuse without relying on elaborate physical access controls or special purpose hardware.

Cryptographic smartcards can be used to implement more carefully controlled and fully revocable file system key escrow. Smartcards have several properties that lend themselves to use as a controlled store for escrowed keys. These cards are designed to be sufficiently tamper-resistant to allow their use in financial applications, have a controlled-access non-volatile memory, can run general purpose software and include built-in cryptographic and random number generation capabilities.

3. Smartcard-Based Key Escrow in a Cryptographic File System

The shortcomings of entirely software-based key escrow schemes arise out of the inability to control the use of the key once it has been revealed to the escrow agent. Thus the problem is to guarantee the escrow agent use of the key without actually revealing what it is. While this may appear to involve impossibly contradictory requirements, most commercial smartcards can be adapted to serve exactly this purpose.

We propose a system in which an "escrow smartcard" can be created along with each file encryption key. This card is provided to a designated third party (the "escrow agent") who is authorized to use the key under some well-defined set of circumstances. If emergency access is required the card can decrypt files without revealing what the key is, acting as a self-contained decryption engine for ciphertext sent to it by the escrow agent. Any time the card decrypts data it also records that fact in its secure storage. Later, when the escrow period is terminated or when an audit is to be performed, the user can query the card to determine whether the escrow agent has used it. This section describes the design and implementation of a smartcard-based key escrow scheme for CFS, a file encryption system for Unix.

CFS is a cryptographic file system interface for Unix-like systems; it allows the user to associate cryptographic keys with directories. It runs entirely on the client workstation. No modification to the underlying file system (or file server) is required, and file contents as well as some meta-data (file names) are cryptographically protected. Backups and other such routine administrative services can take place in the normal manner and without the encryption keys. Details on CFS can be found in [1].

Basically, CFS provides a mechanism to associate "real" directories (on other file systems) that contain encrypted data with temporary "virtual" names through which users can read and write cleartext. These virtual names appear in a separate namespace under the CFS mount point, which is usually called `/crypt`. Users create encrypted directories on regular file systems (e.g., in their home directories) using the **cmkdir** command, which creates the directory and assigns to it a cryptographic "passphrase" that will be used to encrypt its contents. To use an encrypted directory, it must be "attached" to CFS using the **cattach** command, which asks for the passphrase and installs an association between the "real" directory and a name under `/crypt`. Cleartext is read and written under the virtual directory in `/crypt`, but the files are stored in encrypted form (with encrypted names) in the real directory. When the directory is not in use, the association is removed with the **cdetach** command, which deletes the cleartext virtual directory under `/crypt`. When CFS is run on a client workstation, the cleartext data (and the cryptographic key passphrase) are never stored on a disk or sent over a network, even when the real directory is located on a remote file server. The system is implemented as a user-level NFS[11] server. The basic flow of data in CFS is shown in Figure 1.

Key escrow is implemented for CFS as an option to escrow the key when the encrypted directory is created with cmkdir. When keys are initially assigned and whenever escrowed access is required, the machine running CFS must have a smartcard reader-writer attached. (In day-to-day user operation on encrypted files, no smartcard reader is required.) The smartcard has a small store of secure memory, the ability to run simple programs securely and a secret-key cryptographic engine compatible with that of the host file system. Ideally, the card could have a real-time calendar and the ability to schedule execution at some future date, although the cards we use (the AT&T smartcard) do not have these capabilities. We call the user who created the files the "owner" and the caretakers of the escrowed keys the "escrow

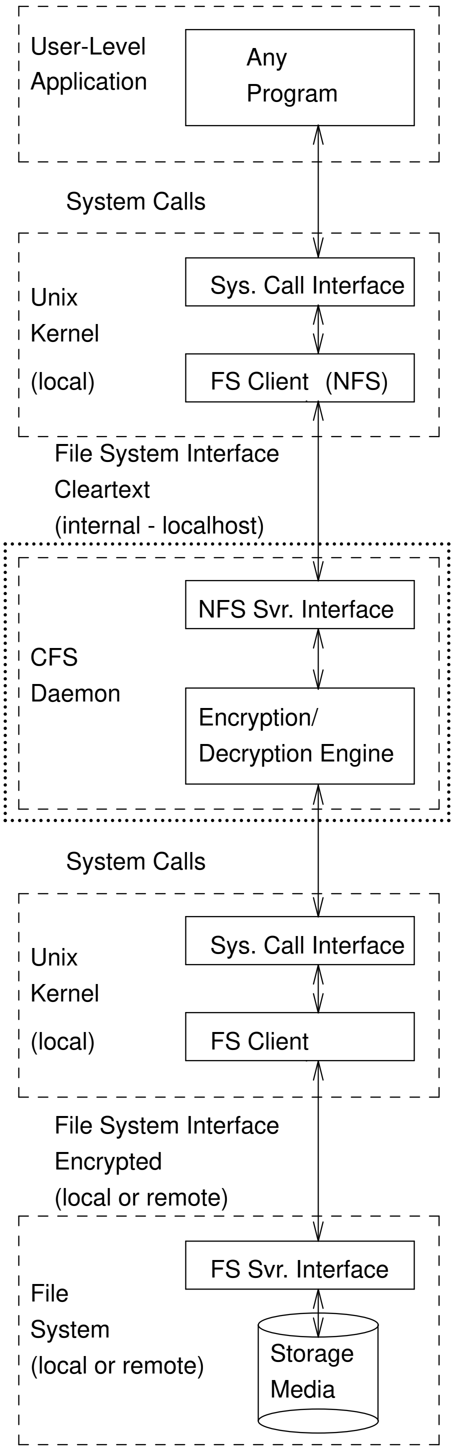

Figure 1 – Data Flow in Standard CFS System

agents." The techniques described here could be applied to any file encryption system and are not limited to CFS.

At the time keys are assigned (e.g., with the CFS cmkdir command), the smartcard is initialized with three sets of cryptographic keys. The first key set, the "file system key," is used for actual file data encryption, and consists, in CFS, of two 56 bit DES[6] keys derived from a user-selected secret "passphrase." The file key is also used to hash a known plaintext string that is stored in the host file system in the "check file." The second key, the "audit key," is used to post-audit the card at escrow revocation time and will be explained in more detail below. The audit key is also stored in a file on the host computer (encrypted under the file keys). The last key, the "escrow key," is used to encrypt the file system keys stored on the card. It must also be provided to the escrow agent (perhaps via public key techniques, and perhaps split among several agents, but this key is not essential to the security of the protocol). Ordinarily, the escrow key is derived from a second passphrase entered by the owner. The encrypted file keys and audit key are maintained in secure storage on the card and cannot be easily "reverse engineered" from the card. All smartcard initialization takes place in CFS through a modified version of the cmkdir command.

Once keys are assigned, the smartcard is turned over to an escrow agent for safekeeping and the escrow key passphrase revealed to the escrow agent. (The escrow agent who holds the card need not be the same agent who knows the escrow key). If the smartcard has a calendar and the ability to schedule future execution, the escrow data on the cards could be configured to automatically self-destruct after a set period. If needed, duplicate cards, with new escrow and audit keys, can be created by the owner (using the file passphrase) at any time.

In normal CFS operation, the file system keys are derived from the user passphrase on the trusted host computer when the owner issues the "cattach" command for an encrypted directory; the smartcard is not involved. Regular user operation requires only the standard version of CFS (without any escrow software). The check file assures that the entered phrase is valid and that wildly incorrect decrypted file names and contents are not returned to the file system.

The smartcard itself is used to perform three operations. The first operation, "pre-audit," simply verifies to the escrow agent that the keys on the card correspond to those used to encrypt the actual file system. In this mode of operation, the escrow agent sends the contents of the check file (in the escrowed file system) and the escrow key to the smartcard, which provides a "yes" or "no" answer based on the decrypted file keys. (The owner could "cheat" and provide a "dummy" check file; we discuss this below.) The escrowed keys do not leave the card.

In "escrow access" operation, the smartcard decrypts files for the escrow agents. The agents supply the escrow key; if it is supplied correctly, the card decrypts the file system keys and increments a counter in its secure store. Thereafter, for the remainder of the session, the card will use the decrypted file keys to decrypt file data sent to it. If the card has a real time clock, it could also maintain two time stamps for the first and most recent times the escrow key was used. Again, the keys never leave the card; the card acts as a wholely self-contained decryption engine. Once the card is removed, its state is reset and the escrow key must be supplied again to enable further decryption. Escrow access in CFS takes place through a modified CFS file system daemon in which the crypto engine is replaced with calls to the smartcard interface. Additional support tools supply the escrow key to the smartcard. Note that the card interface is part of the data path for all decrypted data. The data flow is shown in Figure 2.

The last mode of operation, "post-audit," is used when the escrow period is ended and the card is returned to the owner. The card reports the number of times the escrow keys were used. If the card has the capability to store this data it could also report the first and last access times and number of bytes decrypted under escrow (again, our cards do not). To help protect against card forgery and to safeguard against the return of a fake card by the escrow agent, the owner can challenge the card to perform encryptions under the audit key. The audit key is decrypted on the host computer with the owner passphrase; by comparing the results of a random challenge with the result of a decryption performed locally, the owner can verify that the card that was returned is the same one that was originally escrowed. Post audit is performed in CFS with an additional user tool.

3.1. File Encryption Scheme

One of the lessons learned from the design of CFS is that the problem of encrypting files on-line in a file system is somewhat different from other kinds of encryption problems. No single standard encryption mode[7] has all the properties required for file system use; further compounding the problem are concerns that the 56 bits of key used by the DES cipher are vulnerable to exhaustive search attack[14].

CFS uses a combination of DES "codebook" and pre-computable "stream" cipher modes to approximate the strength of multiple iterations of DES with the runtime latency of only a single iteration of DES. This scheme has the resistance to structural analysis of a chaining cipher but allows random read and write access in constant time.

The encryption scheme relies on the ability to trade off space (in the precomputation of the streams) for time. To accommodate the key escrow system, we modified the CFS encryption scheme to allow "lazy evaluation" on the smartcard without the large memory requirements of the precomputed stream. We believe this scheme to be equivalent to 3-DES under currently known practical attacks. The new CFS cipher is as follows:

Recall that keys in CFS consist of two DES keys, K_1 and K_2, derived from the user passphrase. Conceptually, CFS file block encryption consists of encryption against a positionally defined stream cipher derived from K_1, which is then encrypted with a codebook block cipher under K_2, which is further encrypted with a second multi-use stream cipher derived from K_1. Specifically,

$$E_p = DES^1(K_2, D_p \oplus DES^1(K_1, f(p \ mod \ m)) \oplus i) \oplus DES^1(K_1, g(p \ mod \ m)) \quad (1)$$

The cipher is reversed in the obvious manner:

$$D_p = DES^{-1}(K_2, E_p \oplus DES^1(K_1, g(p \ mod \ m))) \oplus DES^1(K_1, f(p \ mod \ m)) \oplus i \quad (2)$$

where:

E_p is the ciphertext block of a file at byte offset p.

D_p is the cleartext block of a file at byte offset p.

$\oplus$ is the bitwise exclusive-or operation.

$DES^1(k, b)$
is the Data Encryption Standard block encryption function on cleartext b with key k.

$DES^{-1}(k, b)$
is the Data Encryption Standard block decryption function on ciphertext b with key k.

i is a bit representation of a unique file identifier derived from the Unix inode number and creation time of the encrypted file.

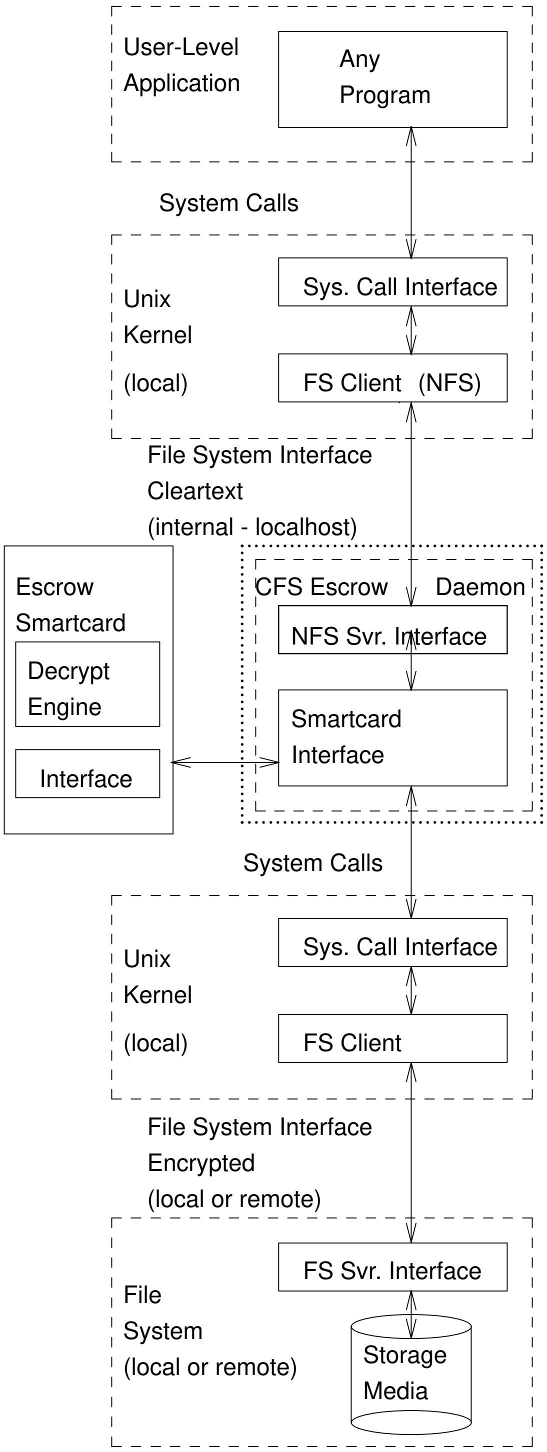

Figure 2 – Data Flow in CFS Escrow Agent System

$f(n), g(n)$
: are publically known functions that map an integer representation n into unique bit strings of the DES codebook size (64 bits).

m
: is the length of the precomputed stored stream (presently 256K bytes).

Observe that the stream ciphers defined by $DES^1(K_1, f(p \ mod \ m))$ and $DES^1(K_1, g(p \ mod \ m))$ can be precomputed for each K_1 given $2m$ bytes of storage. The CFS daemon precomputes these streams when the cattach command is issued for a particular key. With the streams precomputed, each block encryption requires only one online DES operation (the codebook cipher based on K_2).

When decryption is performed on the card, the streams cannot be wholely precomputed in the card's small local memory. Instead, the card calculates $DES^1(K_1, f(p \ mod \ m))$ and $DES^1(K_1, g(p \ mod \ m))$ for each cipherblock sent to it. ($f(p \ mod \ m)$ and $g(p \ mod \ m)$ are sent to the card from the host computer as parameters with the cipher block.) Although this is computationally slower than the precomputed cipher, requiring three DES encryptions per block instead of one, bandwidth to the card interface (a serial link) remains the primary limitation on encryption speed.

4. Practical Applications

File system key escrow can support a variety of application domains. Ensuring organizational access to proprietary data was discussed and motivated above. Here, an employee has primary operational responsibility for data that belongs to an organization. Key escrow allows the organization to provide other individuals with emergency access capability in the primary employee's absence. Access by these "backup" individuals can be granted, controlled, audited and revoked easily, without compromising the organization's ability to maintain and control its own information.

Smartcard-based escrow also facilitates other backup access relationships. In the organizational scenario above, the primary key holder is "subordinate" to the escrow holder. Alternatively, a manager may be the primary key holder for sensitive-but-critical business data for which the keys are escrowed with an employee. The escrow key holder may not be authorized for routine access, but in the manager's absence may be required to perform "proxy" functions on the manager's behalf. Here, the smartcard system implements and enforces a common business delegation of authority practice.

Another scenario, which may become more important in the future, involves the protection of individual personal records. Consider, for example, a system in which medical records are encrypted under a key known only to the patient. Routine use of these records by a health practitioner requires the patient's active consent in supplying the key. In an emergency, however, access to the records may be required even when the patient is physically unable to supply the key. A key escrow smartcard, which might remain in the physical possession of the patient or be maintained with the records themselves, would enable such emergency access but still permit the patient to control (and revoke) the routine use of his or her private records. The proposed US national health care insurance system includes a smartcard-based identification token into which such a scheme could possibly be integrated.

4.1. Performance

The standard CFS system employs a software-based cryptographic engine that performs encryption on a modern workstation at between one and three Mbps[4]. Because CFS uses the standard file system cache, actual performance is much better, with a performance penalty of only 20-50% above the underlying file system under typical workloads. The escrow access system, on the other hand, performs all cryptographic operations on the smartcard, which communicates with the host workstation at serial link speeds (19,200 bps). After protocol and processing overhead, cryptographic bandwidth to the card is about 6,000 bps with the CFS cipher described in the previous section. Using the smartcard for decryption slows the cryptographic engine by almost three orders of magnitude. Cache performance hides this slightly, but the escrow access system is by no means transparent or fast enough for routine operational use.

In practice, the reduced performance is rarely an issue, since escrow access is not intended to support routine processing. (Write operations by the escrow agent are not even supported by our implementation). The normal mode of escrow operation involves copying out those files required for emergency access, such that the card is not subsequently required for their use.

These are not fundamental limitations. Faster smartcards are beginning to emerge in the market, along with faster interfaces with bandwidths that exceed the crypto-bandwidth of current software implementations. PCMCIA cards hold particular promise in this area.

4.2. Trust Model

Smartcard-based key escrow does not absolutely guarantee that the access policy will be enforced. There are risks associated with various parts of the system, each of which must be assessed in light of the application's security policy, threat model and available alternatives.

The system depends on the reverse engineering resistance of the escrow smartcard devices to control access by the escrow agents. Reverse engineering could reveal the keys stored on the card and permit the escrow agent to create duplicate cards without the knowledge of the key owner. Although the risk of reverse engineering is difficult to quantify as technology progresses, commercial smartcards are designed to resist this sort of attack. Recent trends in tamper-resistant packaging and chip fabrication technology suggest the emergence of future products with greatly reduced vulnerability to reverse engineering. In highly sensitive environments in which the integrity of the smartcard is not completely trusted, the card can be protected with augmented physical safeguards such as sealed envelopes and accountable paper audit trails.

By definition, the escrow agent has access to the escrowed data while in possession of the escrow card. The only built-in control on the escrow agent is access detection when the card is eventually audited. If the card is not returned by the agent, however, it is not possible to audit past access or prevent future access as long as the encrypted data remains available. The escrow key serves to limit unauthorized use of lost or stolen cards. If no single agent is trusted, possession of the card and the escrow key can be split among two or more agents. These risks are largely a function of the relationship between the escrow agent and the key owner. When appropriate, the owner can periodically audit the escrow card throughout the escrow period. Controls on access to the encrypted escrowed data can further ameliorate the risk of unauthorized access by the agent.

Any escrow system carries the risk of "cheating" by a key owner who encrypts data with keys other than those escrowed. This risk is present any time the key owner is able to supply his or her own cryptographic system. The check file in the smartcard system guards only against mistakes, not against deliberate deception. All escrow systems suffer from this limitation. In a centralized key distribution system, nothing prevents the use of "out of band" keys not obtained from the key center. In a system such as the government Escrowed Encryption Standard[8] (the "Clipper chip"), it is possible to suppress the escrow exploitation field in the data stream or pre-encrypt with a secure non-escrowed cryptosystem. (The government system attempts to reduce this risk by supplying the escrowed devices in tamper-resistant modules, making it difficult to deploy the cipher without the escrow features.)

The risk of end-user escrow circumvention depends on the relationship between the key owner and the escrow agent. If escrow is perceived as a service for the mutual benefit of the key owner and agent, this risk is not an issue. If, on the other hand, this relationship is adversarial, there can be no completely reliable mechanism that prevents cheating.

5. Conclusions

Key escrow is not appropriate for all file encryption applications. Some data are simply too private; personal diaries, certain individual medical and financial records and other data for which there is no motivation for the data owner to allow third party access are poor candidates for escrow. Other data, such as day-to-day operational business records, have such high availability requirements to preclude any encryption at all. Escrow serves the "middle ground" for which security requirements suggest the need for cryptographic protection while availability requirements dictate the need for access.

Smartcard-based escrow overcomes the major shortcomings of software-based and manual escrow systems. Unlike manual systems, the escrowed keys can be reliably pre-audited to ensure their validity without compromising sensitive data. And unlike either system, once the card is returned, the owner is assured of whether the escrow process was used and that no further decryptions can occur. Escrowed decryption is completely under the control of the card; past possession of the card conveys no future privileges.

6. Acknowledgements

The author is indebted to Doug McIlroy for suggesting the encrypted file access problem. Jim Reeds' critical insights influenced the design of the CFS cipher. Eleanor Evans, Jack Lacy, Tom London and Adam Moskowitz made many helpful suggestions that improved this paper and the system it describes. We are particularly grateful to the anonymous referee who suggested medical records as an application area.

7. Availability

A research prototype of the base CFS system (implemented as a user-level NFS server) is available free upon request within the US and Canada. We regret that US Government-imposed export restrictions prevent us from making it available elsewhere. For information, ftp `dist/mab/cfs.announce` from `research.att.com` or send email to `cfs@research.att.com`. The smartcard software, including the escrow system described here, is not presently available.

8. References

[1] Blaze, M., "A Cryptographic File System for Unix." *Proc. First ACM Conference on Computer and Communications Security,* Fairfax, VA, November 1993.

[2] Denning, D. E., "Encryption and Law Enforcement." *Georgetown University, Computer Science Dept.,* Feb. 21, 1994, available by anonymous ftp from `cpsr.org`.

[3] Diffie, W. and Hellman, M. E., "New Directions in Cryptography." *IEEE Trans. on Information Theory,* IT-11:644-654, November 1976.

[4] Lacy, J., Mitchell, D. and Schell, W., "CryptoLib: Cryptography in Software." *Proc. Fourth USENIX Security Workshop,* October 1993.

[5] Ioannidis, J. and Blaze, M., "Architecture and Implementation of Network-Layer Security Under Unix." *Proc. Fourth USENIX Security Workshop,* October 1993.

[6] National Bureau of Standards, "Data Encryption Standard." *FIPS Publication #46,* NTIS, April 1977.

[7] National Bureau of Standards, "Data Encryption Standard Modes of Operation." *FIPS Publication #81,* NTIS, December 1980.

[8] National Institute for Standards and Technology, "Escrowed Encryption Standard." *FIPS Publication #185,* NTIS, February 1994.

[9] Reiher, P., et. al., "Security Issues in the Truffles File System." *PSRG Workshop on Network and Distributed System Security,* 1993.

[10] Rivest, R.L., Shamir, A. and Adleman, L., "A Method of Obtaining Digital Signatures and Public-Key Cryptosystems." *CACM,* February 1978.

[11] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D. and Lyon, B., "Design and Implementation of the Sun Network File System." *Proc. USENIX,* Summer 1985.

[12] Steiner, J., Neuman, C. and Schiller, J.I., "Kerberos: An Authentication Service for Open Network Systems." *Proc. USENIX,* Winter 1988.

[13] Tygar, J.D. and Yee, B., "Strongbox: A System for Self Securing Programs." *CMU Computer Science: 25th Anniversary Commemorative,* Addison-Wesley, 1991.

[14] Weiner, M.J., "Efficient DES Key Search." *Crypto '93,* (short presentation) August 1993.

Matt Blaze is a member of the Computing Systems Research Laboratory at AT&T Bell Laboratories, which he joined in 1992. His interests include secure systems, large-scale systems, network services, file systems, key management and privacy technology. Matt has a PhD in computer science from Princeton University and is a member of the adjunct faculty at Columbia University. You can reach him via email at `mab@research.att.com` or via postal mail at AT&T Bell Laboratories, 101 Crawfords Corner Rd, Room 4G-634, Holmdel, NJ 07733.