# Cryptographic File Systems Performance: What You Don't Know Can Hurt You

Charles P. Wright, Jay Dave, and Erez Zadok
*Stony Brook University*

**Appears in the proceedings of the 2003 IEEE Security In Storage Workshop (SISW 2003)**

## Abstract

Securing data is more important than ever, yet cryptographic file systems still have not received wide use. One barrier to the adoption of cryptographic file systems is that the performance impact is assumed to be too high, but in fact is largely unknown. In this paper we first survey available cryptographic file systems. Second, we perform a performance comparison of a representative set of the systems, emphasizing multiprogrammed workloads. Third, we discuss interesting and counterintuitive results. We show the overhead of cryptographic file systems can be minimal for many real-world workloads, and suggest potential improvements to existing systems. We have observed not only general trends with each of the cryptographic file systems we compared but also anomalies based on complex interactions with the operating system, disks, CPUs, and ciphers.

**Keywords:** secure storage, performance, cryptographic file systems, stackable file systems, loop devices, user-level file servers

## 1 Introduction

Securing data is more important than ever. As the Internet has become more pervasive, security attacks have grown. Widely-available studies report millions of dollars of lost revenues due to security breaches [23]. There is a wide range of systems and techniques that can ensure data confidentiality. We believe there are three primary factors when evaluating security systems: security, performance, and ease-of-use. These concerns often compete; for example, if a system is too difficult to use, then users simply circumvent it entirely. Furthermore, if users perceive encryption to slow down their work, they just turn it off. Though performance is an important concern when evaluating cryptographic file systems, no rigorous real-world comparison of their performance has been done to date.

Riedel described a taxonomy for evaluating the performance and security that each type of system could theoretically provide [25]. His evaluation encompassed a broad array of choices, but because it was not practical to benchmark so many systems, he only drew theoretical conclusions. In practice, however, deployed systems interact with disks, caches, and a variety of other complex system components — all having a dramatic effect on performance.

In this paper we perform a real world performance comparison between several systems that are used to secure file systems on laptops, workstations, and moderately-sized file servers. We also emphasize multiprogramming workloads, which are not often investigated. Multi-programmed workloads are becoming more important even for single user machines, in which Windowing systems are often used to run multiple applications concurrently. We expect cryptographic file systems to become a commodity component of future operating systems.

We present results from a variety of benchmarks, analyzing the behavior of file systems for metadata operations, raw I/O operations, and combined with CPU intensive tasks. We also use a variety of hardware to ensure that our results are valid on a range of systems: Pentium vs. Itanium, single CPU vs. SMP, and IDE vs. SCSI. We observed general trends with each of the cryptographic file systems we compared. We also investigated anomalies due to complex, counterintuitive interactions with the operating system, disks, CPU, and ciphers. We propose future directions and enhancements to make systems more reliable and predictable.

The rest of this paper is organized as follows. Section 2 surveys existing cryptographic file systems. Section 3 discuss various ciphers used for cryptographic file systems. Section 4 compares the performance of a cross section of cryptographic file systems. We conclude in Section 5.

## 2 File Encryption Systems

This section describes a range of available techniques of encrypting data and is organized by increasing levels of abstraction: block-based systems, native disk file systems, network-based file systems, stackable file systems, and encryption applications.

### 2.1 Block-Based Systems

Block-based encryption systems operate below the file system level, encrypting one disk block at a time. This is advantageous because they do not require knowledge of the file system that resides on top of them, and can even be used for swap partitions or applications that require access to raw partitions (such as database servers). Also, they do not reveal information about individual files (such as sizes and owners) or directory structure.

**Cryptoloop** The Linux loopback device driver presents a file as a block device, optionally transforming the data before it is written and after it is read from the native file, usually to provide encryption. Linux kernels include a cryptographic framework, CryptoAPI [1] that exports a uniform interface for all ciphers and hashes. Presently, IPsec and the Cryptoloop driver use these facilities.

We investigated three backing stores for the loopback driver: (1) a preallocated file created using `dd` filled with random data, (2) a raw device (e.g., `/dev/hda2`), and (3) a sparse backing file created using `truncate(2)`. The left of Figure 1 shows the path that data takes from an application to the file system, when using a raw device as a backing store. The right of Figure 1 shows the path when a file is used as the backing store. The major difference between the two systems is that there is an additional file system between the application and the disk when using a file instead of a device. Using files rather than devices adds performance penalties including cutting the effective buffer cache in half because blocks are stored in memory both as encrypted and unencrypted data. Each of these methods has advantages and disadvantages related to security and ease-of-use as well. Using preallocated files is more secure than using sparse files, because an attacker can not distinguish random data in the file from encrypted data. However, to use a preallocated file, space must be set aside for encrypted files ahead of time. Using a sparse backing store means that space does not need to be preallocated for encrypted data, but reveals more about the structure of the file system (since the attacker knows where and how large the holes are). Using a raw device allows an entire disk or partition to be encrypted, but requires repartitioning, which is more complex than simply creating a file. Typically there are also a limited number of partitions available, so on multi-user systems encrypted raw devices are not as scalable as using files as the backing store.

**CGD** The CryptoGraphic Disk driver, available in NetBSD, is similar to the Linux loopback device, and other loop-device encryption systems, but it uses a native disk or partition as the backing store [9]. CGD has a fully featured suite of user-space configuration utilities that include $n$-factor authentication and PKCS#5 for transforming user passwords into encryption keys [26]. This system is similar to Cryptoloop using a raw device as a backing store.

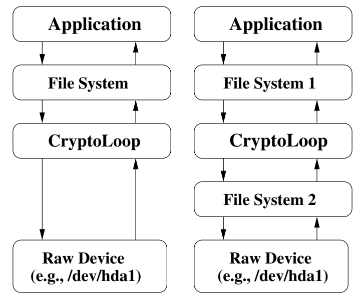


*Figure 1: Cryptoloop stacked on top of a raw device (left) and a file (right)*

**GBDE** GEOM-base disk encryption (GBDE) is based on GEOM, which provides a modular framework to perform transformations ranging from simple geometric displacement for disk partitioning, to RAID algorithms, to cryptographic protection of the stored data. GBDE is a GEOM transform that enables encrypting an entire disk [13]. GBDE hashes a user-supplied passphrase into 512 bits of key material. GBDE uses the key material to locate and encrypt a 2048 bit master key and other metadata on four distinct *lock sectors*. When an individual sector is encrypted, the sector number and bits of the master key are hashed together using MD5 to create a *kkey*. A randomly generated key, the *sector key*, is encrypted with the *kkey*, and then written to disk. Finally, the sector's payload is encrypted with the sector key and written to disk. This technique, though more complex, is similar to Cryptoloop using a raw device as a backing store.

**SFS** SFS is an MSDOS device driver that encrypts an entire partition [11]. SFS is similar to Cryptoloop using a raw device as a backing store. Once encrypted, the driver presents a decrypted view of the encrypted data. This provides the convenient abstraction of a file system, but relying on MSDOS is risky because MSDOS provides none of the protections of a modern operating system.

**BestCrypt** BestCrypt is a commercially available loopback device driver supporting many ciphers [12]. BestCrypt supports both Linux and Windows, and uses a normal file as a backing store (similar to using a preallocated file with Cryptoloop).

**vncrypt** vncrypt is the cryptographic disk driver for FreeBSD, based on the *vn*(4) driver that provides a disk-like interface to a file. vncrypt uses a normal file as a backing store (similarly to using a preallocated file with Cryptoloop) and provides a character device interface, which FreeBSD uses for file systems and swap devices. vncrypt is similar to using Cryptoloop with a file as a backing store.

**OpenBSD vnd** Encrypted file systems support is part of OpenBSD's Vnode Disk Driver, *vnd*(4). *vnd* uses two modes: one bypasses the buffer cache, the second uses the buffer cache. For encrypted devices, the buffer cache must be used to ensure cache coherency on unmount. The only encryption algorithm implemented so far is Blowfish. *vnd* is similar to using Cryptoloop with a file as a backing store.

## 2.2 Disk-Based File Systems

Disk-based file systems that encrypt data are located at a higher level of abstraction than block-based systems. These file systems have access to all per-file and per-directory data, so they can perform more complex authorization and authentication than block-based systems, yet at the same time disk-based file systems can control the physical data layout. This means that disk-based file systems can limit the amount of information revealed to an attacker about file size and owner, though in practice these attributes are often still revealed in order to preserve the file system's on-disk structure. Additionally, since there is no additional layer of indirection, disk-based file systems can have performance benefits over other techniques described in this section (including the loop devices).

**EFS** EFS is the Encryption File System found in Microsoft Windows, based on the NT kernel (Windows 2000 and XP) [18]. It is an extension to NTFS and utilizes Windows authentication methods as well as Windows ACLs [19, 25]. Though EFS is located in the kernel, it is tightly coupled with user-space DLLs to perform encryption and the user-space Local Security Authentication Server for authentication [30]. This prevents EFS from being used for protecting files or folders in the root or `\winnt` directory. Encryption keys are stored on the disk in a *lockbox* that is encrypted using the user's login password. This means that when users change their password, the lockbox must be reencrypted. If an administrator changes the user's password, then all encrypted files become unreadable. Additionally, for compatibility with Windows 2000, EFS uses DESX [28] by default and the only other available cipher is 3DES (included in Windows XP or in the Windows 2000 High Encryption pack).

**StegFS** StegFS is a file system that employs steganography as well as encryption [17]. If adversaries inspect the system, then they only know that there is some hidden data. They do not know the contents or extent of what is hidden. This is achieved via a modified Ext2 kernel driver that keeps a separate block-allocation table per *security level*. It is not possible to determine how many security levels exist without the key to each security level. When the disk is mounted with an unmodified Ext2 driver, random blocks may be overwritten, so data is replicated randomly throughout the disk to avoid loss of data. Although StegFS achieves plausible deniability of data's existence, the performance degradation is a factor of 6–196, making it impractical for most applications.

## 2.3 Network Loopback Based Systems

Network-based file systems (NBFSs) operate at a higher level of abstraction than disk-based file systems, so NBFSs can not control the on-disk layout of files. NBFSs have two major advantages: (1) they can operate on top of any file system, and (2) they are more portable than disk-based file systems. NBFS's major disadvantages are performance and security. Since each request must travel through the network stack, more data copies are required and performance suffers. Security also suffers because NBFS are vulnerable to all of the weaknesses of the underlying network protocol (usually NFS [29, 35]).

**CFS** CFS is a cryptographic file system that is implemented as a user-level NFS server [4]. The cipher and key are specified when encrypted directories are first created. The CFS daemon is responsible for providing the owner with access to the encrypted data via an *attach* command. The daemon, after verifying the user ID and key, creates a directory in the mount point directory that acts as an unencrypted window to the user's encrypted data. Once attached, the user accesses the attached directory like any other directory. CFS is a carefully designed, portable file system with a wide choice of built-in ciphers. Its main problem, however, is performance. Because it runs in user mode, it must perform many context switches and data copies between kernel and user space. As can be seen on the left of Figure 2, since CFS has an unmodified NFS client which communicates with a modified NFS server, it must run only over the loopback network interface, *lo*.

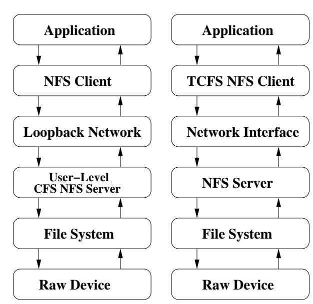


*Figure 2: Call path of network-based systems. CFS is on the left, TCFS is on the right.*

**TCFS** TCFS is a cryptographic file system that is implemented as a modified kernel-mode NFS client. Since it is used in conjunction with an NFS server, TCFS works transparently with the remote file system, eliminating the need for specific attach and detach commands.

To encrypt data, a user sets an encrypted attribute on directories and files within the NFS mount point [6]. TCFS integrates with the UNIX authentication system in lieu of requiring separate passphrases. It uses a database in `/etc/tcfspwdb` to store encrypted user and group keys. Group access to encrypted resources is limited to a subset of the members of a given UNIX group, while allowing for a mechanism (called *threshold secret sharing*) for reconstructing a group key when a member of a group is no longer available. As can be seen on the right of Figure 2, TCFS uses a modified NFS client, which must be implemented in the kernel. This does, however, allow it to operate over any network interface and to work with remote servers.

TCFS has several weaknesses that make it less useful for deployment. First, the reliance on login passwords as user keys is not safe. Also, storing encryption keys on disk in a key database further reduces security. Finally, TCFS is available only on systems with Linux kernel 2.2.17 or earlier, limiting its availability.

### 2.4 Stackable File Systems

Stackable file systems are a compromise between kernel-level disk-based file systems and loopback network file systems. Stackable file systems can operate on top of any file system; they do not have to copy data across the user-kernel boundary or through the network stack; and they are portable to several operating systems [38].

**Cryptfs** Cryptfs is the stackable, cryptographic file system and part of the FiST toolkit [37]. Cryptfs was never designed to be a secure file system, but rather a proof-of-concept application of FiST [38]. Cryptfs supports only one cipher and implements a limited key management scheme. Cryptfs serves as the basis for several commercial and research systems (e.g., ZIA [7]). Figure 3 shows Cryptfs's operation. A user application invokes a system call through the Virtual File System (VFS). The VFS calls the stackable file system, which again invokes the VFS after encrypting or decrypting the data. The VFS calls the lower level file system, which writes the data to its backing store.

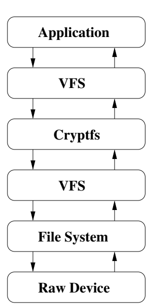


*Figure 3: Call path of stackable file systems.*

**NCryptfs** NCryptfs is our stackable cryptographic file system, designed with the explicit aim of balancing security, convenience, and performance [34]. NCryptfs allows system administrators and users to customize NCryptfs according to their specific needs. NCryptfs supports multiple concurrent authentication methods, multiple dynamically-loadable ciphers, ad-hoc groups, and challenge-response authentication. Keys, active sessions, and authorizations in NCryptfs all have timeouts. NCryptfs can be configured to transparently suspend and resume processes based on key, session or authorization validity. NCryptfs also enhanced the kernel to discard cleartext pages and notify the file system on process exit in order to expunge invalid authentication entries.

### 2.5 Applications

File encryption can be performed by applications such as GPG [16] or crypt(1) that reside above the file system. This solution, however, is quite inconvenient for users. Each time they want to access a file, users must manually decrypt or encrypt it. The more user interaction is required to encrypt or decrypt files, the more often mistakes are made, resulting in damage to the file or leaking confidential data [33]. Additionally, the file may reside in cleartext on disk while the user is actively working on it.

File encryption can also be integrated into each application (e.g., text editors or mail clients), but this shifts the burden from users to applications programmers. Often applications developers do not believe the extra effort of implementing features is justified when only a small fraction of users would take advantage of those features. Even if encryption is deemed an important enough feature to be integrated into most applications there are still two major problems with this approach. First, each additional application that the user must trust to function correctly reduces the overall security of the system. Second, since each application may implement encryption slightly differently it would make using files in separate programs more difficult.

## 3 Ciphers

For cryptographic file systems there are several ciphers that may be used, but those of interest are generally symmetric block ciphers. This is because block ciphers are efficient and versatile. We discuss DES variants, Blowfish, and Rijndael because they are often used for file encryption, and are believed to be secure. There are many other block ciphers available, including CAST, GOST, IDEA, MARS, Serpent, RC5, RC6, and TwoFish. Most of them have similar characteristics with varying block and key sizes.

**DES** DES is a block cipher designed by IBM with assistance from the NSA in the 1970s [28]. DES was the first encryption algorithm that was published as a standard by NIST with enough details to be implemented in software. DES uses a 56-bit key, a 64-bit block size, and can be implemented efficiently in hardware. DES is no longer considered to be secure. There are several more secure variants of DES, most commonly 3DES [20]. 3DES uses three separate DES encryptions with three different keys, increasing the total key length to 168 bits. 3DES is considered secure for government communications. DESX is a variant designed by RSA Data Security that uses a second 64-bit key for *whitening* (obscuring) the data before the first round and after the last round of DES proper, thereby reducing its vulnerability to brute-force attacks, as well as differential and linear cryptanalysis [28].

**Blowfish** Blowfish is a block cipher designed by Bruce Schneier with a 64-bit block size and key sizes up to 448 bits [28]. Blowfish had four design criteria: speed, compact memory usage, simple operations, and variable security. Blowfish works best when the key does not change often, as is the case with file encryption, because the key setup routines require 521 iterations of Blowfish encryption. Blowfish is widely used for file encryption.

**AES (Rijndael)** AES is the successor to DES, selected by a public competition. Though all of the six finalists were judged to be sufficiently secure for AES, the final choice for AES was Rijndael based on the composite of the three selection criteria (security, cost, and algorithm characteristics) [21]. Rijndael is a block cipher based on the Square cipher that uses S-boxes (substitution), shifting, and XOR to encrypt 128-bit blocks of data. Rijndael supports 128, 192, and 256 bit keys.

# 4 Performance Comparison

To compare the performance of cryptographic systems we chose one system from each category in Section 2, because we were interested in evaluating properties of techniques, not specific systems. We chose benchmarks that measure file system operations, raw I/O operations, and a simulated user workload. Throughout the comparisons we ran multi-programmed tests to compare increasingly parallel workloads and scalability.

We did not benchmark an encryption application, because other programs can not transparently access encrypted data. As representatives from each category we chose Cryptoloop, EFS, CFS, and NCryptfs. We chose Cryptoloop, EFS, and CFS because they are widely used, and up-to-date. Cryptoloop can be run on top of raw devices and normal files, which makes it comparable to a wider range of block-based systems than a block-based system that uses only files or only block devices. We chose NCryptfs over Cryptfs because NCryptfs is more secure. We also tried to choose systems that run on a single operating system, so that operating system effects would be consistent. We used Linux for most systems, but for disk-based file systems we chose Windows. There was no suitable and widely-used disk-based solution for Linux, large part because block-based systems are generally used on Linux. From previous studies, we also knew that TCFS and BestCrypt had performance problems and we therefore omitted them [34].

We chose workloads that stressed file system and I/O operations, particularly when there are multiple users. We did not use a compile benchmark or other similar workloads, because they do not effectively measure file system performance under heavy loads [31]. In Section 4.1 we describe our experimental setup. In Section 4.2 we report on PostMark, a benchmark that exercises file system meta-data operations such as lookup, create, delete, and append. In Section 4.3 we report on PGMeter, a benchmark that exercises raw I/O throughput. In Section 4.4 we report on AIM Suite VII, a benchmark that simulates large multiuser workloads. Finally, in Section 4.5 we report other interesting results.

## 4.1 Experimental Setup

To provide a basis for comparison we also perform benchmarks on Ext2 and NTFS. Ext2 is the baseline for Cryptoloop, CFS, and NCryptfs. NTFS is the baseline for EFS. We chose Ext2 rather than other Linux file systems because it is widely used and well-tested. We chose Ext2 rather than the Ext3 journaling file system, because the loopback device's interaction with Ext3 has the effect of disabling journaling. For performance reasons, the default journaling mode of Ext3 does not journal data, but rather only journals meta-data. When a loopback device is used with a backing store in an Ext3 file, the Ext3 file system contained inside the file does journal meta-data, but it writes the journal to the lower-level Ext3 file as data, which is not journaled. Additionally, journaling file systems create more complicated I/O effects by writing to the journal as well as normal file system data.

We used the following configurations:

- A vanilla Ext2 file system.
- A loopback device using a raw partition as a backing store. Ext2 is used inside the loop device. We refer to this configuration as LOOPDEV.
- A loopback device using a preallocated backing store. Both the underlying file system and the file system contained within the loop device are Ext2. We refer to this configuration as LOOPDD.
- CFS using an Ext2 file system as a backing store.
- A vanilla NTFS file system.
- An encrypted folder within an NTFS file system

(EFS).

- NCryptfs stacked on top of an Ext2 file system.

**System Setup** For Cryptoloop and NCryptfs we used four ciphers: the Null (identity) transformation to demonstrate the overhead of the technique without cryptography; Blowfish with a 128-bit key to demonstrate overhead with a cipher commonly used in file systems; AES with a 128-bit key because AES is the successor to DES [8]; and 3DES because it is an often-used cipher and it is the only cipher that all of the tested systems supported. We chose to use 128-bit keys for Blowfish and AES, because that is the default key size for many systems. For CFS we used Null, Blowfish, and 3DES (we did not use AES because CFS only supports 64-bit ciphers). For EFS we used only 3DES, since the only other available cipher is DESX.

In Table 1 we summarize various system features that affect performance.

**1. Category** LOOPDEV and LOOPDD use a block-based approach. EFS is implemented as an extension to the NTFS disk-based file system. CFS is implemented as a user-space NFS server. NCryptfs is a stackable file system.

**2. Location** LOOPDEV, LOOPDD, and NCryptfs are implemented in the kernel. EFS uses a hybrid approach, relying on user-space DLLs for some cryptographic functions. CFS is implemented in user space.

**3. Buffering** LOOPDEV and EFS keep only one copy of the data in memory. LOOPDD, CFS, and NCryptfs have both encrypted and decrypted data in memory. Double buffering effectively cuts the buffer cache size in half.

**4. Encryption unit** LOOPDEV, LOOPDD, and EFS use a disk block as their unit of encryption. This defaults to 512 bytes for our tested systems. CFS uses the cipher block size as its encryption unit and only supports ciphers with 64-bit blocks. NCryptfs uses the PAGE_SIZE as the unit of encryption: on the i386 this is 4KB and on the Itanium this defaults to 16KB.

**5. Encryption mode** LOOPDEV and LOOPDD use CBC encryption. The public literature does not state what mode of encryption EFS uses, though it is most probably CBC mode because: (1) the Microsoft CryptoAPI uses CBC mode by default, and (2) it has a fixed block size that is accommodating to CBC. NCryptfs uses *cipher text stealing* (CTS) to encrypt data of arbitrary length to data of the same length.

CFS does not use standard cipher modes, but rather a hybrid of ECB and OFB. Chaining modes (e.g., CBC) do not allow direct random access to files, because to read or write from the middle of a file all the previous data must be decrypted or encrypted first. However, ECB mode permits a cryptanalyst to do structural analysis. CFS solves this by doing the following: when an attach is created, half a megabyte of pseudo-random data is generated using OFB mode and written to a *mask file*. Before data is written to a data file, it is XORed with the contents of the mask file at the same offset as the data (modulo the size of the mask file), then encrypted with ECB mode. To read data this process is simply reversed. This method is used to allow uniform access time to any portion of the file while preventing structural analysis. NCryptfs and Cryptoloop achieve the same effect using initialization vectors (IVs) and CBC on pages and blocks rather than the whole file.

For Cryptoloop, CFS, and NCryptfs we adapted the Blowfish, AES, and 3DES implementations from OpenSSL 0.9.7b to minimize effects of different cipher implementations [24]. Cryptoloop uses CBC mode encryption for all operations. NCryptfs uses CTS to ensure that an arbitrary length buffer can be encrypted to a buffer of the same length [28]. NCryptfs uses CTS because size changing algorithms add complexity and overhead to stackable file systems [36]. CTS mode differs from CBC mode only for the last two blocks of data. For buffers that are shorter than the block size of the cipher, NCryptfs uses CFB, since CTS requires at least one full block of data. We did not change the mode of encryption that CFS uses.

**6. Write Mode** LOOPDD does not use a synchronous file as its backing store, therefore all writes become asynchronous. CFS uses NFSv2 where all writes on the server must be synchronous, but CFS violates the specification by opening all files asynchronously. Due to VFS calling conventions, NCryptfs does not cause the lower-level file system to use write synchronously. Asynchronous writes improve performance, but at the expense of reliability. NCryptfs uses a write-through strategy: whenever a `write` system call is issued, NCryptfs passes it to the lower-level file system. The lower-level file system may not do any I/O, but NCryptfs still encrypts data.

**Testbed** We ran our benchmarks on two machines: the first represents a workstation or a small work group file server, and the second represents a mid-range file server.

The workstation machine is a 1.7Ghz Pentium IV with 512MB of RAM. In this configuration all experiments took place on a 20GB 7200 RPM Western Digital Caviar IDE disk. For Cryptoloop, CFS, and NCryptfs we used Red Hat Linux 9 with a vanilla Linux 2.4.21 kernel. For EFS we used Windows XP (Service pack 1) with high encryption enabled for EFS.

Our file server machine is a 2 CPU 900Mhz Itanium 2 McKinley (hereafter we refer to this CPU as an Itanium) with 8GB of RAM running Red Hat Linux 2.1 Advanced Server with a vanilla SMP Linux 2.4.21 kernel. All ex-

|   | Feature | LOOPDEV | LOOPDD | EFS | CFS | NCryptfs |
|---|---|---|---|---|---|---|
| 1 | **Category** | Block Based | Block Based | Disk FS | NFS | Stackable FS |
| 2 | **Location** | Kernel | Kernel | Hybrid | User Space | Kernel |
| 3 | **Buffering** | Single | Double | Single | Double | Double |
| 4 | **Encryption unit** | 512B | 512B | 512B | 8B | 4KB/16KB |
| 5 | **Encryption mode** | CBC | CBC | CBC (?) | OFB+ECB | CTS/CFB |
| 6 | **Write Mode** | Sync,Async | Async Only | Sync,Async | Async Only | Async only, Write-Through |

*Table 1: Features related to performance*

periments took place on a Seagate 73GB 10,000 RPM Cheetah Ultra160 SCSI disk. Only Ext2, Cryptoloop, and NCryptfs were tested on this configuration. We did not test CFS because its file handle and encryption code are not 64-bit safe. We did not test NTFS or EFS because 64-bit Windows is not yet commonly used on the Itanium platform.

All tests were performed on a cold cache, achieved by unmounting and remounting the file systems between iterations. The tests were located on a dedicated partition in the outer sectors of the disk to minimize ZCAV and other I/O effects [10].

For multi-process tests we report the elapsed time as the maximum elapsed time of all processes, which shows how long it takes to actually get the allotted work completed. We report the system time as the sum of the system times of each process and kernel thread involved in the test. For LOOPDD we add the CPU time of the kernel thread used by the loopback device to perform encryption. For LOOPDEV we add in the CPU time used by `kupdated`, because encryption takes place when syncing dirty buffers. Finally, for CFS we add the user and system time used by `cfsd`, because this is CPU time used on behalf of the process.

As expected there were no significant variations in the user time of the benchmark tools, because no changes took place in the user code. We do not report user times for these tests because we do not vary the amount of work done in the user process.

We ran all tests several times, and we report instances where our computed standard deviations were more than 5% of the mean. Throughout this paper, if two values are within 5%, we do not consider that a significant difference.

## 4.2 PostMark

PostMark focuses on stressing the file system by performing a series of file system operations such as directory lookups, creations, appends, and deletions on small files. A large number of small files is common in electronic mail and news servers where multiple users are randomly modifying small files. We configured PostMark to create 20,000 files and perform 200,000 transactions in 200 subdirectories. To simulate many concurrent users, we ran each configuration with 1, 2, 4, 8, 16, and 32 concurrent PostMark processes. The total number of transactions, initial files, and subdirectories was divided evenly among each process. We chose the above parameters for the number of files and transactions as they are typically used and recommended for file system benchmarks [14, 32]. We used many subdirectories for each process, so that the work could be divided without causing the number of entries in each directory to affect the results (Ext2 uses a linear directory structure). We ran each test at least ten times.

Through this test we demonstrate the overhead of file system operations for each system. First we discuss the results on our workstation configuration in Section 4.2.1. Next, we discuss the results on our file server configuration in Section 4.2.2.

### 4.2.1 Workstation Results

**Ext2** Figure 4 shows elapsed time results for Ext2. For a single process, Ext2 ran for 121.5 seconds, used 12.2 seconds of system time, and average CPU utilization was 32.1%. When a second process was added, elapsed time dropped to 37.1 seconds, less than half of the original time. The change in system time was negligible at 12.5 seconds. System time remaining relatively constant is to be expected because the total amount of work remains fixed. The average CPU utilization was 72.1%. For four processes the elapsed time was 17.4 seconds, system time was 9.9 seconds, and CPU utilization was 79.4%. After this point the improvement levels off as the CPU and disk became saturated with requests.

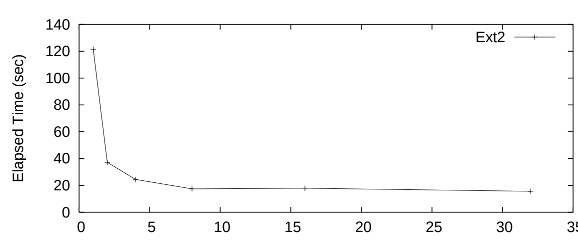


*Figure 4: PostMark: Ext2 elapsed time for the workstation*

**LOOPDEV** Figure 5 shows elapsed time results for LOOPDEV, which follows the same general trend as Ext2. The single process test is dominated by I/O and subsequent tests improve substantially, until the CPU is saturated. A single process took 126.5 seconds for Null, 106.7 seconds for Blowfish, 111.8 seconds for AES, and 136.3 seconds for 3DES. The overhead over Ext2 is 4.2% for Null, -12.2% for Blowfish, -8.0% for AES, and 12.25% for 3DES. The fact that Blowfish and AES are faster than Ext2 is an artifact of the disks we used. When we ran the experiment on a ramdisk, a SCSI disk, or a slower IDE disk, the results were as expected. Ext2 was fastest, followed by Null, Blowfish, AES, and 3DES. The system times, shown in Figure 6, were 12.4, 13.5, 14.0, and 27.4 seconds for Null, Blowfish, AES and 3DES, respectively. The average CPU utilization was 31.6%, 38.3%, 36.6%, 40.0% for Null, Blowfish, AES and 3DES, respectively. When there were eight concurrent processes, the elapsed times were 17.3, 18.4, 18.8, and 22.8 seconds, respectively. The overheads were 1.0–32.2% for eight processes. The average CPU utilization ranged from 64.1–80.1%. The decline in system time was unexpected, and is due to the decreased elapsed time. Because dirty buffers have a shorter lifetime when elapsed time decreases, `kupdated` does not flush the short-lived buffers, thereby reducing the amount of system time spent during the test.

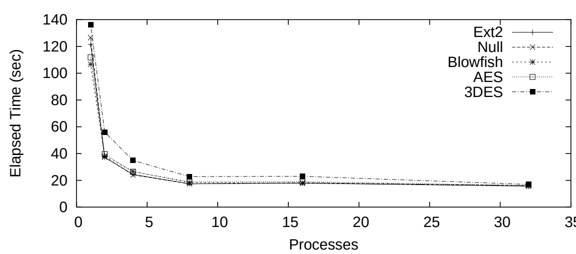


*Figure 5: PostMark:* LOOPDEV *elapsed time for the workstation*

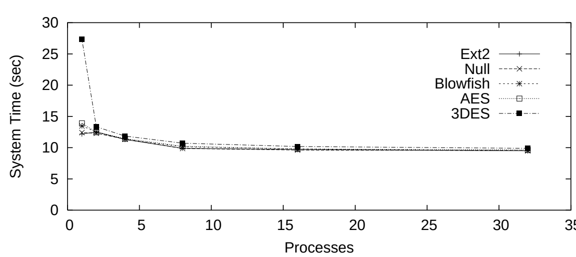


*Figure 6: PostMark:* LOOPDEV *system time for the workstation*

The results fit the pattern established by Ext2 in that once the CPU becomes saturated, the elapsed time remains constant. Again, the largest benefit is seen when going from one to two processes. Encryption does not have a large user visible impact, even for this intense workload.

**LOOPDD** Figure 7 shows the elapsed time results for the LOOPDD configuration. The elapsed times for a single process are 41.4 seconds for Null, 44.5 for Blowfish, 45.7 for AES, and 73.9 for 3DES. For eight processes, the elapsed times decrease to 23.1, 23.5, 24.5, and 50.39 seconds for Null, Blowfish, AES, and 3DES, respectively.

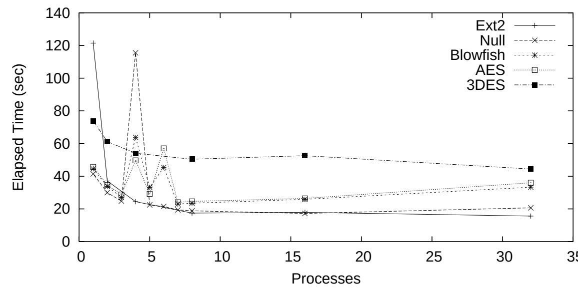


*Figure 7: PostMark:* LOOPDD *elapsed time for the workstation. Note: Null, Blowfish, and AES additionally have points for 1–8 in one process increments rather than exponentially increasing processes.*

With LOOPDD we noticed an anomaly at four processes. We have repeated this test over 50 times and the standard deviation remains high at 40% of the mean for Null, AES, and Blowfish. There is also an inversion between the elapsed time and the cipher speed. The Null cipher is the slowest, followed by Blowfish, and then AES. 3DES is not affected by this anomaly. We have investigated this anomaly in several ways. We know this to be an I/O effect, because the system time remains constant and when the test is run with a ram disk, the anomaly disappears. We also ran the test for the surrounding points at three, five, six, and seven concurrent processes. For Null there were no additional anomalies, but for Blowfish and AES the results for 4–6 process were erratic and standard deviations ranged from 25–67%. We have determined the anomaly's cause to be an interaction with the Linux buffer flushing daemon, `bdflush`. Flushing dirty buffers in Linux is controlled by three parameters: $nfract$, $nfract\_stop$, and $nfract\_sync$. Each parameter is a percentage of the total number of buffers available on the system. When a buffer is marked dirty, the system checks if the number of dirty buffers exceeds $nfract\%$ (by default 30%) of the total number of buffers. If so, `bdflush` is woken up and begins to sync buffers until only $nfract\_stop\%$ of the system buffers are dirty (by default 20%). After waking up `bdflush`, the kernel checks if more than $nfract\_sync\%$ of the total buffers are dirty (by default 60%). If so, then the process synchronously flushes `NRSYNC` buffers (hard coded to 32) before returning control to the process. The $nfract\_sync$ is designed to throttle heavy writers and ensure that enough clean buffers are available. We changed the $nfract\_sync$ parameter to 90% and reran this test. Figure 8 shows the results when using $nfract\_sync = 90\%$ and the anomaly is gone. Because the Null processes are writing to the disk so quickly, they end up causing the number of dirty buffers to go over the $nfract\_sync$ threshold. The four process test is very close to this threshold, which accounts for the high standard deviation. When more CPU is used, either through more processes or slower ciphers, the rate of writes is slowed, and this this effect is not encountered. We have confirmed our analysis by increasing the RAM on the machine from 512MB to 1024MB, and again the elapsed time anomaly disappeared.

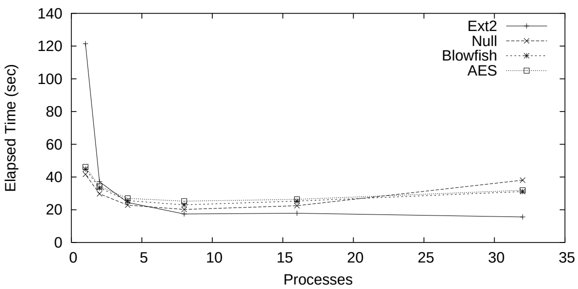


*Figure 8: PostMark:* LOOPDEV *elapsed time for the workstation, with $nfract\_sync = 90\%$. Note: 3DES is not included because it does nto display this anomaly.*

The other major difference between LOOPDD and LOOPDEV is that LOOPDD uses more system time. This is for two reasons. First, LOOPDD traverses the file system code twice, once for the test file system and once for the file system containing the backing store. Second, LOOPDD effectively cuts the buffer and page caches in half by double buffering. Cutting the buffer cache in half means that fewer cached cleartext pages are available so more encryption operations must take place. The system time used for LOOPDD is also relatively constant, regardless of how many processes are used. We instrumented a CryptoAPI cipher to count the number of bytes encrypted, and determined that unlike LOOPDEV, the number of encryptions and decryptions does not significantly change. The LOOPDEV system marks the buffers dirty and issues an I/O request to write that buffer. If another write comes through before the I/O is completed, then the writes are coalesced into a single I/O operation. When LOOPDD writes a buffer it adds the buffer to the end of a queue for the loop thread to write. If the same buffer is written twice, the buffer is added to the queue twice, and hence encrypted twice. This prevents searching through the queue, and since the lower-level file system may in fact coalesce the writes, this does not have a large impact on elapsed time.

The results show that LOOPDD systems have several complex interactions with many components of the system that are difficult to explain or predict. When maximal performance and predictability is a consideration, LOOPDEV should be used instead of LOOPDD.

**CFS** Figure 9 shows CFS elapsed times, which are relatively constant no matter how many processes are running. Since CFS is a single threaded NFS server, this result was expected. System time also remained constant for each test: 146.7–165.3 seconds for Null, 298.9–309.4 seconds for Blowfish, and 505.7–527.8 seconds for 3DES. There is a decrease in elapsed time when there are eight concurrent processes (27.0% for Null, 14.1% for Blowfish, and 7.8% for 3DES), but the results return to their normal progression for 16 processes. The dip at eight processes occurs because I/O time decreases. At this point the request stream to our particular disk optimally interleaves with CPU usage. System time remains the same. When this test is run inside of a ram disk or on different hardware, this anomaly disappears.

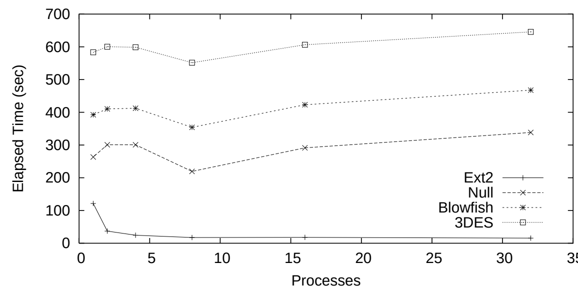


*Figure 9: PostMark: CFS elapsed time for the workstation*

We conclude that both the user-space and single-threaded architecture are bottlenecks for CFS. The single threaded architecture prevents CFS from making use of parallelism, while the user-space architecture causes CFS to consume more system time for data copies to and from user space and through the network stack.

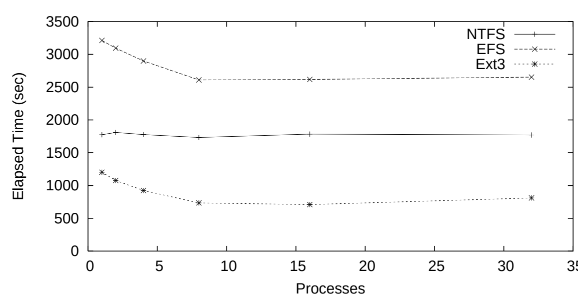


*Figure 10: PostMark: NTFS elapsed time for the workstation*

**NTFS** Figure 10 shows that NTFS performs significantly worse than Ext2/3 for the PostMark benchmark. The elapsed time for NTFS was relatively constant regardless of the number of processes, ranging from 28.9–30.2 minutes. Katcher reported that NTFS performs poorly for this workload when there are many concurrent files [14]. Unlike Ext2, NTFS is a journaling file system. To compare NTFS with a similar file system we ran this benchmark for Ext3 with journaling enabled for both meta-data and data. Ext2 took 2.1 minutes for one process, and Ext3 took 20.0 minutes. We hypothesize that the journaling behavior of NTFS negatively impacts its performance for PostMark operations. NTFS has many more synchronous writes than Ext3, and all writes have three phases that must be synchronous. First, changes to the meta-data are written to allow a read to check for corruption or crashes. Second, the actual data is written. Third, the metadata is fixed, again with a synchronous write. Between the first and last step, the metadata may not be read or written to, because it is in effect invalid. Furthermore, the on-disk layout of NTFS is more prone to seeking than that of Ext2 (Ext3 has an identical layout). NTFS stores most metadata information in the *master file table* (MFT), which is placed in the first 12% of the partition. Ext2 has group descriptors, block bitmaps, inode bitmaps, and inode tables for each cylinder group. This means that a file's meta-data is physically closer to its data in Ext2 than in NTFS.

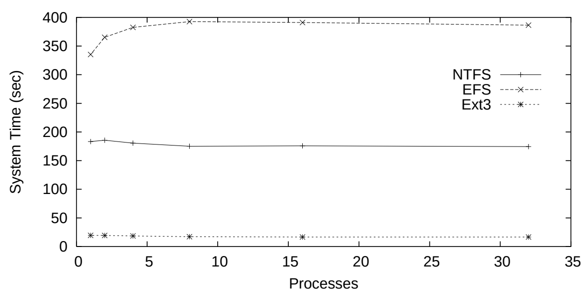


*Figure 11: PostMark: NTFS system time for the workstation*

When an encrypted folder is used for a single process, elapsed time increases to 57.2 minutes, a 98% overhead over NTFS. Figure 11 shows that for a single process, system time increases to 335.2 seconds, a 82.9% overhead over NTFS. We have observed that unlike NTFS, EFS is able to benefit from parallelism when multiple processes are used under this load. When there are eight concurrent processes, the elapsed time decreases to 43.5 minutes, and the system time increases to 386.5 seconds. No additional throughput is achieved after eight processes. We conclude that NTFS, and hence EFS, is not suitable for busy workloads with a large number of concurrent files.

**NCryptfs** Figure 12 shows that for a single process running under NCryptfs, the elapsed times were 136.2, 150.0, 156.5, and 368.9 seconds for Null, Blowfish, AES, and 3DES, respectively. System times were 22.2, 44.0, 54.3, and 287.5 seconds for Null, Blowfish, AES, and 3DES, respectively. This is higher than for LOOPDD and LOOPDEV because NCryptfs uses write-through, so all writes cause an encryption operation to take place. The system time is lower than in CFS because CFS must copy data to and from user-space, and through the network stack. Average CPU utilization was 36.4% for Null, 47.5% for Blowfish, 52.2% for AES, and 84.9% for 3DES. For eight processes, elapsed time was 27.0, 48.2, 56.2, and 281.5 seconds for Null, Blowfish, AES, and 3DES, respectively. CPU utilization was 90.2%, 96.3%, 98.0%, and 99.3% for Null, Blowfish, AES, and 3DES, respectively. This high average CPU utilization indicates that CPU is the bottleneck, preventing greater throughput from being achieved.

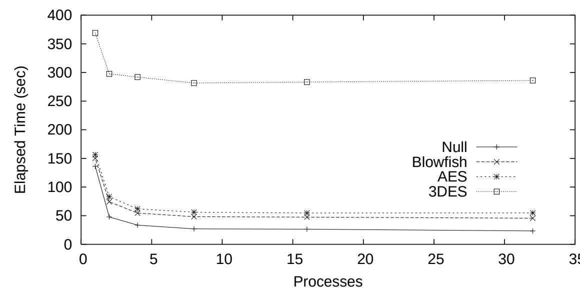


*Figure 12: PostMark: NCryptfs Workstation elapsed time*

**Ciphers** As expected throughout these tests, 3DES introduces significantly more overhead than AES or Blowfish. Blowfish was designed with speed in mind, and a requirement for AES was that it would be faster than 3DES. In this test AES and Blowfish perform similarly.

We chose to use Blowfish for further benchmarks because it is often used for file system cryptography and has properties that are useful for cryptographic file systems. First, Blowfish supports long key lengths, up to 448-bits. This is useful for long-term storage because it is necessary to protect against future attacks. Second, Blowfish generates tables to speed the actual encryption operation. Key setup is an expensive operation, which takes 4168 bytes of memory and requires 521 iterations of the Blowfish algorithm [28]. Since key setup is done infrequently, this does not place undue burden on the user, but makes it somewhat more difficult to mount a brute-force attack.

### 4.2.2 File Server Results

**Single CPU Itanium** Figure 13 shows the results when we ran the PostMark benchmark on our Itanium file server configuration with one CPU enabled (we used the same SMP kernel as we did with two CPUS). The trend is similar to that seen for the workstation. For Ext2, the elapsed time for a single process was 79.9 seconds, and the system time was 11.7 seconds. For LOOPDEV, the elapsed time is within 1% of Ext2 for all tests while for LOOPDD the elapsed times decreased by 31% for all tests compared to Ext2. For NCryptfs the elapsed time was 102.4 seconds for Null and 181.5 seconds for Blowfish. This entails a 24% overhead for Null and 127% overhead for Blowfish over Ext2. Of note is that NCryptfs uses significantly more CPU time than on the i386. The increase in CPU time is caused by the combination of two factors. First, the Blowfish implementation we used (from OpenSSL 0.9.7b) only encrypts at 31.5MB/s on this machine as opposed to 52.6MB/s on the i386. Second, the i386 architecture uses a 4KB page size by default, whereas the Itanium architecture supports page sizes from 4KB–4GB. The Linux kernel uses a 16KB page size by default. Since NCryptfs encodes data in units of one page, any time a page is updated, NCryptfs must re-encrypt more data. When using a Linux kernel recompiled with a 4KB page size on the Itanium, the system time used by NCryptfs with Blowfish drops by 37.5% to 55.6 seconds. With a 4KB page size, the system time that Blowfish uses over Null on the Itanium is roughly the additional time that the i386 uses, scaled by the encryption speed. Since this test is mostly contained in main memory, NCryptfs also performs significantly more work than does the loopback device in this test. NCryptfs uses write-through to the lower-level file system, so each write operation results in an encryption. The loopback device does not encrypt or decrypt data for operations that are served out of the cache.

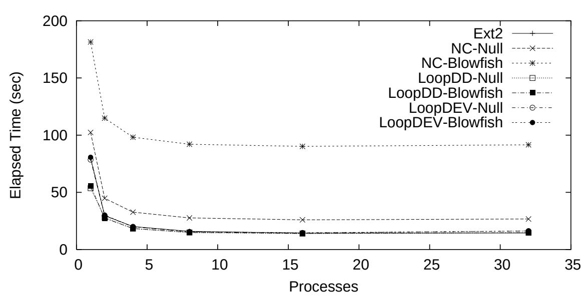


*Figure 13: PostMark: File server elapsed time with one CPU*

We conclude that the general trends for the Itanium architecture are the same as for the i386 architecture. The i386 has had years of investment in optimizing compilers and optimized code, whereas the Itanium being a new architecture has not yet developed an optimized codebase.

**Dual CPU Itanium** Figure 14 shows the results when a second CPU is added to the file server. As expected, the elapsed times for a single process remained unchanged. Again, the bottleneck was the CPU in this experiment, and the additional CPU resources proved to be of benefit. As the number of processes increases past one process, the system time increased slightly for all systems because of the added overhead of locking and synchronization of shared directories and other file-system

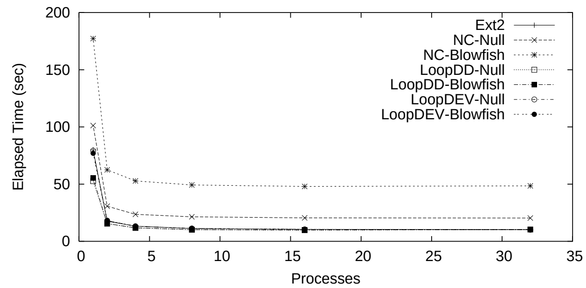


*Figure 14: PostMark: File server elapsed time with two CPUs*

data structures. However, because the increased CPU time was spread across the two CPUs, elapsed time decreased significantly. The decrease in the elapsed times for Ext2 ranged from 27–30% for eight or more concurrently running processes. LOOPDD had similar behavior for both Blowfish and Null ciphers. For LOOPDEV the decrease ranged from 28–37% for both Blowfish and Null ciphers. The elapsed times for NCryptfs changed to 53% of their value (compared to a single CPU) for Blowfish while the decrease for the Null cipher was 23%.

We conclude that these file systems scale as well as Ext2, and when encryption is added, the benefit is even more significant.

## 4.3 PGMeter

PenguinoMeter is a file-I/O benchmark with a workload specification patterned after that of the IOMeter benchmark available from Intel [2, 5]. PGMeter measures the data transfer rate to and from a single file. We ran PGMeter with the `fileserver.icf` workload distributed with PGMeter and IOMeter, which is a mixture of 512B–64KB transfers of which 80% are read requests and 20% are write requests. Each operation begins at a randomly selected point in the file. We ran this workload for 1–256 outstanding I/O requests, increasing the number by a factor of 4 on each subsequent run.

Each test starts with a 30 second ramp-up period in order to avoid transient startup effects, and then performs operations for 10 minutes. Every 30 seconds during the test, PGMeter reports the number of I/O operations performed and the number of megabytes of data transferred per second. We ran each test three times, and observed stable results both across and within tests. Since the amount of data transferred is a multiple of the I/O operations, we do not include this result.

For the workstation configuration the data file size was 1GB, and for the file server it was 16GB. This is twice the amount of memory on the machine, in order to measure I/O performance and not performance when reading or writing to the cache.

On Windows we used version 1.3.22-1 of the Cygwin environment to build and run PGMeter. When given the same specification, PGMeter produces similar workloads as IOMeter [5]. A major limitation with PGMeter and Windows is that Cygwin does not support more than 63 concurrent threads, so a large number of outstanding operations can not be generated. We used IOMeter on Windows with the same workload as PGMeter on Linux. This allowed to compare NTFS and EFS for an increasing number of concurrent operations. Though IOMeter and PGMeter are quite similar, conclusions should not be drawn between NTFS and EFS vs. their Linux counterparts.

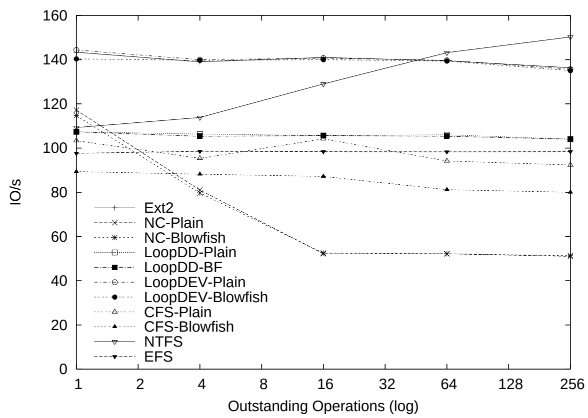


*Figure 15: PGMeter: Workstation I/O operations per second*

Figure 15 shows the operations per second (ops/sec) for PGMeter and IOMeter. As in previous tests we adopted Ext2 as the baseline for the Linux systems. The average I/O ops/sec for Ext2 started at 143 ops/sec for one outstanding operation (each outstanding operation uses a thread on Linux) and dropped to 136.3 ops/sec for 256 threads. The ops/sec for LOOPDEV with Null ranged from 144 ops/sec for one thread to 135.7 ops/sec for 256 threads. When using Blowfish, ops/sec ranged from 140–135 ops/sec. LOOPDD dropped to 107 ops/sec for one thread and 104 ops/sec for 256 threads. This is because LOOPDD must pass through the file system hierarchy twice. The results for Blowfish were identical, indicating that the encryption overhead is not significant. IOMeter with the file server workload under NTFS showed a performance improvement when the number of outstanding operations was increased. For one outstanding operation, 83.7 ops/sec were performed, increasing to 100 ops/sec for 256 outstanding operations. EFS followed suite increasing from 78.7 to 91.7 ops/sec. From this we conclude that even though NTFS performs poorly with many concurrent files, it is able to effectively interleave CPU and disk resources when there is only one open file.

As on the other Linux-based systems, CFS showed the same pattern of a decrease in the ops/sec as the number of threads increased. The range for Null was 107.6 ops/sec for one thread to 92.4 ops/sec for 256 threads. For CFS with Blowfish, the number of ops/sec ranged from 89.3 for one thread to 80 for 256 threads. NCryptfs performed better than LOOPDD and CFS for a single thread, achieving 117 ops/per second for Null and 114 for Blowfish. However, for multiple threads, NCryptfs with Null quickly dropped to 81 ops/sec for 4 threads and 52.2 ops/sec for 16 threads where it began to stabilize. NCryptfs with Blowfish followed the same pattern, stabilizing at 52.1 ops/sec for 16 threads. The reason that NCryptfs performed poorly for multiple threads is that it must down the inode semaphore for each write operation, so they become serialized and there is significant contention. This behavior was not exhibited during PostMark since each process worked on a disjoint set of files.

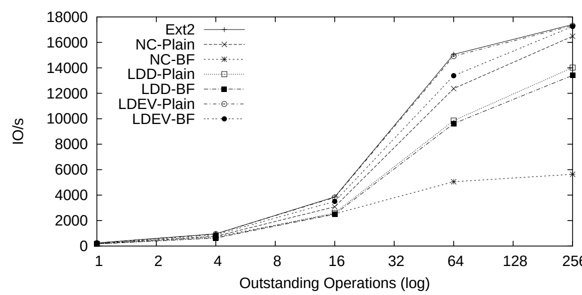


*Figure 16: PGMeter: File server I/O operations per second*

Figure 16 shows the ops/sec achieved on our Itanium fileserver machine. On this machine, we observed an improvement in the ops/sec with an increase in the number of threads due to the interleaving of CPU bound and I/O bound tasks. The number of ops/sec for Ext2 ranged from 244 for one thread to 17,404 for 256 threads. LOOPDEV with Null closely followed Ext2 with 242–17,314 ops/sec. LOOPDEV with Blowfish started off at 220 ops/sec for one thread and reached 17,247 ops/sec for 256 threads. LOOPDD performed fewer ops/sec for both Null and Blowfish, because it had to travel through the file system stack twice. Additionally, LOOPDD cuts the buffer cache in half so fewer requests are served from memory. LOOPDD with Null performed 166–14,021 ops/sec. With Blowfish, the number of ops/sec ranged from 160–13,408. NCryptfs with Null started at 193.3 ops/sec for one thread and increased to 16,482 ops/sec for 256 threads. When using Blowfish, NCryptfs only achieved 176–5,638 ops/sec, due to the computation overhead of Blowfish combined with the write-through implementation, and the larger 16KB default (PAGE_SIZE) unit of encryption.

From this benchmark we conclude that I/O intensive workloads suffer a greater penalty than meta-data intensive workloads on cryptographic file systems. There are several reasons for this: double buffering cuts page caches in half, data must be copied through multiple file systems, and there is increased lock contention. Additionally, the increased unit of encryption begins to have a negative effect on the systems. Though Blaze's hybrid OFB/ECB approach does not suffer from increasing page or block sizes, its security not well studied.

## 4.4 AIM Suite VII

The AIM Multiuser Benchmark Suite VII (AIM7) is a multifaceted system-level benchmark that gives an estimate of the overall system throughput under an ever-increasing workload. AIM7 comes with preconfigured standard workload mixes for multiuser environments. An AIM7 run comprises of a series of sub-runs with the number of tasks increasing in each sub-run. AIM7 creates one process for each simulated operation load (task). Each sub-run waits for all children to complete their share of operations. We can specify the number of tasks to start with and the increment for each subsequent subrun. The AIM7 run stops after it has reached the *crossover* point that indicates the multitasking operation load where the system's performance becomes unacceptable, i.e., less than 1 job/minute/task. Each run takes over 12 hours and is self-stabilizing, so we report the values from only a single run. In our other trials have observed the results to be stable over multiple runs. AIM7 runs a total of 100 tests for each task, selecting the tests based on weights defined by the workload configuration. We chose the fileserver configuration that represents an environment with a heavy concentration of file system operations and integer computations. The file server configuration consists of 40% asynchronous file operations (directory lookup, random read, copy, random writes, and sequential writes), 58% miscelleneous tests that do not depend on the underlying file system and 2.5% synchronous file operations (random write, sequential write and copy). AIM7 reports the total jobs/minute and jobs/minute/task for each subrun. For a typical system, as the number of concurrently running tasks increases, the jobs/minute increases to a peak value as idle CPU time is used; thereafter it reduces due to software and hardware limitations. We ran AIM7 with one initial task and an increment of two tasks for each subrun. AIM7 uses an *adaptive timer* to determine the increment after the first eight subruns. After the invocation of the adaptive timer the number of tasks depends on the recent rate of change of throughput, but the adaptive timer often significantly overshoots the crossover point [3]. AIM7 can also use Newton's method to approach the crossover point more gradually. We opted to use Newton's method.

The results of AIM7 can be seen in Figure 17. Ext2 served as a base line for all the tests. The peak jobs/minute on Ext2 was 230 with 5 tasks and the crossover point was 112 tasks. LOOPDEV with a Null

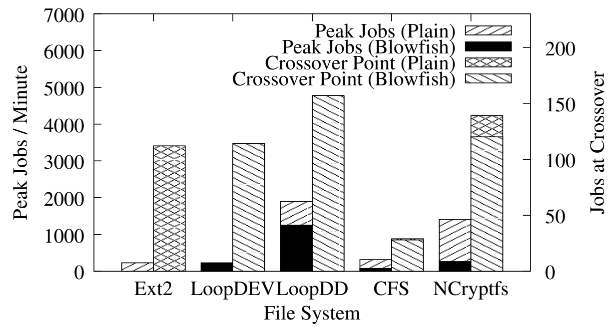


*Figure 17: AIM7 results for the workstation. Note: Peak jobs/minute is using the left bars and scale. Crossover is using the right bars and scale.*

cipher peaked at 229 jobs/minute with 5 tasks and reached crossover at 108 tasks. With Blowfish, the peak jobs/minute was 217 with 5 tasks and the crossover was at 114 tasks. LOOPDD, NCryptfs, and CFS all achieved a higher peak jobs/minute and crossover point than Ext2. This is because they do not respect synchronous write requests. When run with only asynchronous writes, Ext2 peaks at 2253 jobs/minute. LOOPDD peaked at 1897 jobs/minute with 46 tasks using Null; and 1255 jobs/minute with 40 tasks using Blowfish. The crossover point was 156 tasks for Null, and 157 tasks for Blowfish. This difference is not significant. NCryptfs with Null peaked at 1405 jobs/minute with 34 tasks, and reached the crossover at 139 tasks. With Blowfish, NCryptfs peaked at 263 jobs/minute with 23 tasks, and reached the crossover at 120 tasks. When using Blowfish with NCryptfs, the write-through behavior causes CPU utilization to increases and the integer computation portion of the AIM7 test is not able to achieve as high a throughput. CFS with Null peaked at only 317 jobs/minute with 11 tasks, despite the fact that it does not respect the synchronous flag. This is because there is a significant amount of overhead incurred by traversing the network stack and copying data to and from user space. The crossover point was 29 tasks. When CFS is used with Blowfish, the peak throughput is reduced to 77 jobs/minute with 9 tasks. The crossover is reached at 28 tasks.

From these results we conclude that synchronous I/O may be a small part of most workloads, but it has an overwhelming effect on the total throughput. We also conclude that as workloads shift to an intensive mix of I/O and CPU usage, encryption begins to affect throughput more throughput. Our previous experience shows that less intense workloads, such as compiles, have little user visible overhead [34].

On the Itanium file servers, Ext2 reached a peak jobs/sec count of 188 for 93 tasks and reached crossover at 171 tasks. LOOPDEV again closely mimicked the behavior of Ext2 with both Null and Blowfish reaching a peak of 171 jobs/minute with 103 tasks for Null and 117 tasks for Blowfish. The crossover points were 166 and 165 for Null and Blowfish, respectively. If only the asynchronous I/O portions are executed, Ext2 has a peak throughput of 4524. As compilers for the Itanium improve, the computational portions of this suite of tests should also improve to exceed the throughput of the i386.

LOOPDD with Null and Blowfish showed the same trend as the workstation machines. With Null, the peak jobs/min was 3315 with 48 tasks, and the crossover point 934 tasks. Blowfish peaked at 1613 jobs/min with 200 tasks, and the crossover point was 1049. Because the Blowfish implementation is slower on the Itanium, the peak jobs/min dropped by 51% when compared with Null. On the i386 the peak jobs/min only dropped by 34%. NCryptfs with Null peaked at 2365 jobs/min with 167 tasks and the crossover point was 1114 tasks. With Blowfish, NCryptfs becomes significantly slower, peaking at 132 jobs/min with 132 tasks and reaching crossover at 133 tasks. This can again be attributed to the interference with CPU-bound processes, and is only exacerbated by compilers not yet optimizing the Blowfish implementation as well as on the i386.

From this we conclude that even though the raw I/O performance is significantly better on the Itanium, a mix of heavily compute and I/O intensive workloads still suffer from the slower CPU. When using a cryptographic file system with write-through this is only worsened. We expect that as Itanium compilers improve, this problem will be less pronounced.

## 4.5 Other Benchmarks

In this section we explain interesting results involving configurations not previously discussed.

**Key Size** A common assumption is that stronger security decreases performance. A classic example of this is that if you use longer key lengths, then encryption is more costly. We set out to quantify exactly how much it would cost to increase your key length to the maximum allowed by our ciphers. We used two variable key length ciphers: Blowfish and AES.

| Cipher | Key Size (bits) | Speed MB/s |
|---|---|---|
| AES | 128 | 27.5 |
| AES | 192 | 24.4 |
| AES | 256 | 22.0 |
| Blowfish | Any | 52.6 |
| 3DES | 168 | 10.4 |
| Null | None | 694.7 |

*Table 2: Workstation raw encryption speeds*

To encrypt data, most block ciphers repeat a simple function several times; each time the primitive function is iterated is called a *round*. Blowfish has 16 rounds no matter what key size you choose; only the key expansion changes, which occurs only on initialization. This means that encryption time remains constant without consideration to whether a 32 or 448 bit key is chosen. Our experimental results confirmed this. No significant difference was recorded for raw encryption speeds for NCryptfs running the PostMark benchmark. On the other hand, AES has 10, 12, or 14 rounds depending on the key size (128, 192, and 256 bits, respectively). Table 2 shows raw encryption speeds on our workstation configuration. There is a 24.7% difference between the fastest and slowest AES key length, but when benchmarked in the context of file systems, this translates into a mere 1.3% difference for NCryptfs running PostMark. This difference is within the margin of error for these tests (here, standard deviations were 1–2% of the mean).

The overhead of encryption is largely masked by I/O and other components of the system time, such as the VFS code. Since the key size plays a small role in the file system throughput, it is wise to use as large a key size as possible. This is made only more important by the persistent nature of file systems, where there is often lots of data available to a cryptanalyst — some of which may be predictable, e.g., partition tables [27]. Since file systems are designed for long-term storage, it is important that data is protected not only against adversaries of today, but also of the future.

**Sparse Loopback Devices** Throughout our previous tests, LOOPDD used a preallocated file that filled the test partition to reduce I/O effects. It is often the case that an encrypted volume is only a small fraction of the disk. This leads to interesting results with respect to I/O. Ext2 attempts to put blocks from the same file into a single cylinder group; and spread directories evenly across cylinder groups with the goal of clustering related files [22].

Using a preallocated file as large as the partition does not drastically affect the Ext2 allocation policy. However, if the preallocated file is smaller than the partition, it is clustered into as few cylinder groups as possible, thereby reducing the amount of seeking required. Using a sparse file as the backing store results in an even more optimal allocation policy than the small file, since holes in the upper-level file system are not allocated.

We ran this test on our workstation with a 10GB Seagate 5400 RPM IDE hard drive. We used a slower disk than in previous tests to accentuate the differences between these configurations. Using the entire drive formatted with Ext2, PostMark took 551 seconds and used 12.2 seconds of system time. When using a 500MB LOOPDD file, Null mode took 79.6 seconds, and used 27.1 seconds of system time. Using Cryptoloop with a sparse backing store took 51.2 seconds and used 27.2 seconds of system time.

From this test we conclude that although using a loop device increases system overhead, the effects that it has on layout policy can be beneficial. On the other hand, the fact that LOOPDD can so markedly affect the layout policies of the file system may have negative impacts for some workloads. One advantage of systems like LOOPDEV is that they will not affect the layout decisions made by the upper level file system.

## 5 Conclusions and Future Work

The contributions of this work are that we have presented a survey of available file encryption systems and we have performed a comprehensive performance comparison of five representative systems: Cryptoloop using a device as a backing store, Cryptoloop using a file as a backing store, EFS, CFS, and NCryptfs.

From this study we draw five conclusions about cryptographic file systems performance. First, it is often suggested that entire file systems should not be encrypted because of performance penalties. We believe that encrypting an entire hard disk is practical. Systems such as Cryptoloop that can encrypt an entire partition are able to make effective use of the buffer cache in order to prevent many encryption and decryption operations.

Second, for single process workloads, I/O is the limiting factor, but the encryption operations lie along a critical path in the I/O subsystem. Since I/O operations can not complete until the encryption/decryption is done, encryption negatively impacts performance in some cases.

Third, the loop device was able to scale along with the underlying file system for both metadata and I/O-intensive operations. NCryptfs was able to scale along with the file system for metadata operations, but not for I/O-intensive operations. The single threaded nature of CFS limits its scalability. EFS and NTFS did not perform well for the workload with many concurrent files, but EFS was able to exploit concurrent processes in both the PostMark and IOMeter test.

Fourth, the effect of caching can not be underestimated when comparing cryptographic file systems. Caches not only decrease the number of I/O operations, but they also avoid costly encryption operations. Even though Blowfish is almost twice as fast as AES on our machines, their results were quite similar, in large part because many I/O requests are served out of the cache.

Fifth, when heavily interleaving disk and CPU operations unexpected effects often occur. Furthermore, these effects are not necessarily gradual degradation of performance, but rather large spikes in the graph. Adjusting any number of things (memory, disk speed, or other kernel parameters) can expose or hide such effects. For performance-critical applications, the specific hardware, operating system, and file system must be benchmarked with both the expected workload as well as lighter and heavier workloads.

Based on our study, we believe block devices are a promising way to encrypt data for installations that do not need the advanced features that can only be provided by a file system (e.g., ad-hoc groups or per-file permissions). However, there are still two major problems with the state of block-based encryption systems. First, loop devices using a file as a backing store are not currently opened with `O_SYNC`. This means that even though the file system contained in the encrypted volume flushes data to the block device, the underlying file system does not flush the data. This interaction increases performance (even when compared to the native file system) at the expense of reliability and integrity constraints. This should be changed in future releases of Cryptoloop. NCryptfs and CFS also do not respect the `O_SYNC` flag, but for different reasons. NCryptfs does not because of the VFS call path, and CFS simply uses all asynchronous I/O.

Second, there are a large number of block-based encryption systems, for the same and different operating systems, but they are not compatible due to different block sizes, IV calculation methods, key derivation techniques, and disk layouts. For these systems to gain more wide-spread use they should support a common on-disk format. A common format would be particularly useful for removable media (e.g., USB drives).

Stackable or disk-based file systems are useful when more advanced features are required. The following three issues should be addressed. (1) The virtual file system should include a centralized, stacking-aware cache manager to allow stackable file systems to inform the lower level system that caching this data is not useful. Eliminating double-buffering would decrease encryption and I/O operations. (2) The write-through implementation used by FiST results in more encryption operations than are required. A write-back implementation would allow writes to be coalesced in the upper-level file system. (3) The encoding size of NCryptfs should be smaller than PAGE_SIZE in order to reduce computational overhead.

**Future Work** Confidentiality can currently be protected through cryptographic file systems, but most systems do not yet properly ensure the integrity of data. Both a file system or loopback device that stores cryptographic checksums of data would greatly increase the overall usefulness of cryptographic file systems. We will investigate the performance impact of checksumming, and the most efficient place to store these checksums (interspersed with data or in a separate location).

Throughout this work we measured throughput, although it is often a secondary concern when compared with latency, which more greatly affects interactivity. As it stands, unencrypted I/O operations can have a large effect on interactivity. Quantifying how cryptographic file systems affect interactivity is an important next step toward evaluating these systems for desktop use.

Hardware encryption cards are becoming more widespread for accelerating SSL connections. Hardware cryptographic accelerators may also increase performance for file systems [15]. For applications such as SSL, where all of the data that is being encrypted is new, there is little opportunity for caching as there is with file systems. We plan to examine how hardware cryptographic accelerators interact with cryptographic file systems and caches under a variety of workloads.

## 6 Acknowledgments

We would like to thank Anton Altaparmakov and Szakacsits Szabolcs for their help in understanding the behavior of NTFS. This work was partially made possible by an NSF CAREER award EIA-0133589, and HP/Intel gifts numbers 87128 and 88415.1.

## References

[1] The GNU/Linux CryptoAPI site. www.kerneli.org, August 2003.

[2] IOMeter. http://iometer.sourceforge.net, August 2003.

[3] AIM Technology. AIM Multiuser Benchmark - Suite VII Version 1.1. http://sourceforge.net/projects/aimbench, 2001.

[4] M. Blaze. A cryptographic file system for Unix. In *Proceedings of the first ACM Conference on Computer and Communications Security*, 1993.

[5] R. Bryant, D. Raddatz, and R. Sunshine. PenguinoMeter: A New File-I/O Benchmark for Linux. In *Proceedings of the 5th Annual Linux Showcase & Conference*, pages 5–10, Oakland, CA, November 2001.

[6] G. Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano. The Design and Implementation of a Transparent Cryptographic Filesystem for UNIX. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 245–252, June 2001.

[7] M. Corner and B. D. Noble. Zero-interaction authentication. In *The Eigth ACM Conference on Mobile Computing and Networking*, September 2002.

[8] Department of Commerce: National Institute of Standards and Technology. Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard. Technical Report Docket No. 960924272-6272-01, January 1997.

[9] R. Dowdeswell and J. Ioannidis. The CryptoGraphic Disk Driver. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, June 2003.

[10] D. Ellard and M. Seltzer. NFS Tricks and Benchmarking Traps. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 101–114, June 2003.

[11] P. C. Gutmann. Secure filesystem (SFS) for DOS/Windows. www.cs.auckland.ac.nz/˜pgut001/sfs/index.html, 1994.

[12] Jetico, Inc. BestCrypt software home page. www.jetico.com, 2002.

[13] P. H. Kamp. *gdbe(4)*, October 2002. FreeBSD Kernel Interfaces Manual, Section 4.

[14] J. Katcher. PostMark: a New Filesystem Benchmark. Technical Report TR3022, Network Appliance. www.netapp.com/tech˙library/3022.html.

[15] A. Keromytis, J. Wright, and T. de Raadt. The Design of the OpenBSD Cryptographic Framework. In *Proceedings of the Annual USENIX Technical Conference*, pages 181–196, June 2003.

[16] W. Koch. The GNU privacy guard. www.gnupg.org, August 2003.

[17] A. D. McDonald and M. G. Kuhn. StegFS: A Steganographic File System for Linux. In *Information Hiding*, pages 462–477, 1999.

[18] Microsoft Corporation. Encrypting File System for Windows 2000. Technical report, July 1999. www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp.

[19] R. Nagar. *Windows NT File System Internals: A developer's Guide*, pages 615–67. O'Reilly, September 1997. Section: Filter Drivers.

[20] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 1999.

[21] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback. Report on the Development of the Advanced Encryption Standard (AES). Technical report, Department of Commerce: National Institute of Standards and Technology, October 2000.

[22] J. Nugent, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Controlling Your PLACE in the File System with Gray-box Techniques. In *Proceedings of the Annual USENIX Technical Conference*, pages 311–323, June 2003.

[23] R. Power. Computer Crime and Security Survey. *Computer Security Institute*, VIII(1):1–24, 2002. www.gocsi.com/press/20020407.html.

[24] The OpenSSL Project. Openssl: The open source toolkit for SSL/TLS. www.openssl.org, April 2003.

[25] E. Riedel, M. Kallahalla, and R. Swaminathan. A Framework for Evaluating Storage System Security. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 15–30, Monterey, CA, January 2002.

[26] RSA Laboratories. Password-Based Cryptography Standard. Technical Report PKCS #5, RSA Data Security, March 1999.

[27] J. H. Saltzer. Hazards of file encryption. Technical report, 1981. http://web.mit.edu/afs/athena.mit.edu/user/other/a/Saltzer/www/publications/csrrfc208.html.

[28] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2 edition, October 1995.

[29] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS Version 4 Protocol. Technical Report RFC 3010, Network Working Group, December 2000.

[30] D. A. Solomon and M. E. Russinovich. *Inside Microsoft Windows 2000*, chapter 12: File Systems, pages 683–778. Microsoft Press, 2000.

[31] D. Tang and M. Seltzer. Lies, Damned Lies, and File System Benchmarks. Technical Report TR-34-94, Harvard University, December 1994. In VINO: The 1994 Fall Harvest.

[32] VERITAS Software. Veritas file server edition performance brief: A postmark 1.11 benchmark comparison. Technical report. http://eval.veritas.com/webfiles/docs/fsedition-postmark.pdf.

[33] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the Eigth Usenix Security Symposium*, August 1999.

[34] C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, June 2003.

[35] E. Zadok. *Linux NFS and Automounter Administration*. Sybex, Inc., May 2001.

[36] E. Zadok, J. M. Anderson, I. Bădulescu, and J. Nieh. Fast Indexing: Support for size-changing algorithms in stackable file systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 289–304, June 2001.

[37] E. Zadok, I. Bădulescu, and A. Shender. Cryptfs: A stackable vnode level encryption file system. Technical Report CUCS-021-98, Computer Science Department, Columbia University, June 1998.

[38] E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 55–70, June 2000.