UNIVERSITY OF OSLO
Department of Informatics

Comparison of automated, user-transparent encryption facilities under Linux and Windows operating systems

Master Thesis

Beibei Zhan
Oslo University College

May 26, 2010

Comparison of automated, user-transparent encryption facilities under Linux and Windows operating systems

Master Thesis

Beibei Zhan
Oslo University College

May 26, 2010

Abstract

Cybercrime is a serious social problem. Data security threats affect not only large financial organizations and government department but also personal computer users. It is very common that laptops or USB drivers are stolen or lost, and private information is leaked as a consequence. Encryption is one method used to ensure data security. During recent years, encryption technology has developed significantly, and there are many encryption facilities available.

Even though users do care about their data security, normally they think it is too sophisticated and time-consuming to install and configure encryption facilities. They also worry about encryption would affect their system's performance and other features. As a result, many people do not even try to use the encryption technologies. Others have tried some encryption tools but didn't like them and then give up. However, users still have the problem of information leakage now.

In this thesis, the author performs research on encryption technologies, describes the features and advantages or disadvantages of the most popular encryption facilities, and also measures and compares the performance penalty of these facilities. Furthermore, recommendations are given based on different aspects of users' requirements. Hopefully, the result of this thesis will be valuable to users who want to use encryption technology and must choose the most suitable facility.

Acknowledgements

First and foremost, I want to express the greatest gratitude to my beloved supervisor Æleen Frisch. Many thanks to her teaching, guidance, encouragement, patience and support during the whole project progress. She sets a very good example for me, as an experienced system administrator as well as a learned professor. I feel very proud and lucky to have been her student and under her supervising. It will be the most unforgettable experience in my life.

I also very appreciate the help from Kyrre M. Begnum, Hårek Haugerud, Simen Hagen and all the other teachers, thank you so much for all your support and care during my master study. I am so proud to be a student in Oslo University College and so happy to work with all my classmates in the last two years.

Last but not least, I want to give special thanks to my family for all the support and encouragement.

Once again, thank you all!

Oslo, May 2010
Beibei Zhan

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Nowadays, information security is very important, organizations of all sizes that are challenged to protect valuable digital information against careless mishandling and malicious attacks. Attacks can take many forms, such as USB abuse, illegal internet access, virus intrusion and so on. It is a demanding task to protect their data not only from random hackers but also against directed attacks from determined attackers.

Solutions like Firewalls, IDS and Access Control strategies, etc, function well at blocking information security threats from network. However, data corruption and information leaks still happen quite often. Rethinking the root cause for those information security threats, lead to the realization that what we really want to protect is the storage media! Valuable data, executable programs, configuration and authorization information, and even the base executable version of the operating system itself are all stored on the storage media, which sometimes must be protected at the source.

Encryption is a common way of protecting data on disk, one method is file-based encryption, which means encrypting the contents of a file, or part of the contents of a file. However, this method still exist encryption attack cracks. Other methods are filesystem level encryption and block device encryption which encrypt the data on a lower level. These encryption methods can encrypt files at the filesystem level or block device, and protect confidential data from attackers with physical access to the computer. Both also give ordinary users transparent access to the encrypted data. Transparency is critical to implementing encryption successfully and delivering peace of mind to managers, administrators, and employees who depend on well-protected data. Users are generally not sophisticated enough to manually encrypt and decrypt files, and they will resist any solution which is inconvenient in any way, so the best solution is the one which they don't even know about. By transparently encrypting a file system or block device, the encryption and decryption operations are performed at a layer below user file access, and the encryption is transparent to the user and all to their applications. In this way, we can achieve both data security and ease at use for ordinary users.

There are many approaches to filesystem encryption and block device encryption currently. In this project, I will describe and compare the features and advantages or disadvantages of the most important approaches, and also perform representative tasks and standard benchmark softwares with and without encryption on different platforms, to evaluate the benefits of these approaches to the normal user.

1.2 Problem Statement

What are the benefits of different encryption methods to normal user?

- Which encryption facility is best for the normal user?
- Is it easy to install and use?
- How big is the performance penalty?
- How do these results vary between the Linux and Windows environments?

I will install and use different encryption tools of filesystem level encryption and block device encryption. I will compare the performance of representative tasks with and without encryption for each encryption tool and platform. For the Windows operating system, I will use BitLocker, EFS, TrueCrypt. For the Linux operating system use eCryptfs, Dm-crypt, TrueCrypt.

Based on the results of these experiments, I will discuss and evaluate the various solutions with respect to the dimensions such as effectiveness, ease of installation and use, and penalty to hard disk performance.

1.3 Thesis Outline

This thesis will be structured as follows:

Chapter 1 introduces about the motivations and goals of this thesis. Readers are introduced to the problem of encryption that the thesis will investigate.

Chapter 2 provides background information relevant to encryption in general and available encryption tools.

Chapter 3 explains the design and setup process of the performance tests. It also discusses installation of the various encryption tools.

Chapter 4 presents the data and results achieved in comparing different encryption approaches' performances. Performance tests are deployed from several aspects.

Chapter 5 discusses about the results achieved, estimating the performance penalty for different encryption approaches, and evaluates the benefits of these tools. Some recommendations for ordinary users are also included. The final sections discuss potential future work and provides conclusion to the thesis.

Chapter 2

Background and Literature

This chapter provides a general overview of data security and then briefly introduces several technologies and terminologies related to encryption. It also outlines the different encryption techniques and tools to be studied.

2.1 Data Security Overview

Cybercrime is not a new word or concept any more. According to a United States official involved in targeting online crime, cybercrime is becoming more organized. In 2006, the FBI estimates all types of computer crime in the US costs industry about $400 billion, while in Britain the Department of Trade and Industry said computer crime had risen by 50 percent over the last two years [1]. During recent years, we also heard many news reports about cybercrime and information leakage incidents.

On October 18, 2008, the German's newspaper FianzNachrichten reported a story entitled "Axel Springer hit by new German data leak scandal." It described how thousands of people's personal data related to placing for classified advertisements in newspapers and millions of mobile phone customers' data from the Deutsche Telekom company was published on the Internet [2].

Similar things continue to happen. On November 17, 2009, the BBC reported that the staff at mobile phone company T-Mobile sold millions of records from thousands of customers to third party brokers [3]. This caused heavy losses to T-Mobile. Also, on November 21, 2009, the Washington Post published another report that described the information leakage in the world's foremost climate research centers, which gave rise to a controversy between climate scientists and climate-change skeptics [4].

In 2008, the Web Application Security Consortium (WASC) announced the results of WASC Web Application Security Statistics Project [5]. The initiative of the project is a collaborative industry-wide effort to pool together sanitized website vulnerability data and to gain a better understanding about the web application vulnerability landscape. In the project, statistics from 12186 sites with 97554 detected vulnerabilities were analyzed. Figure 2.1 shows the results

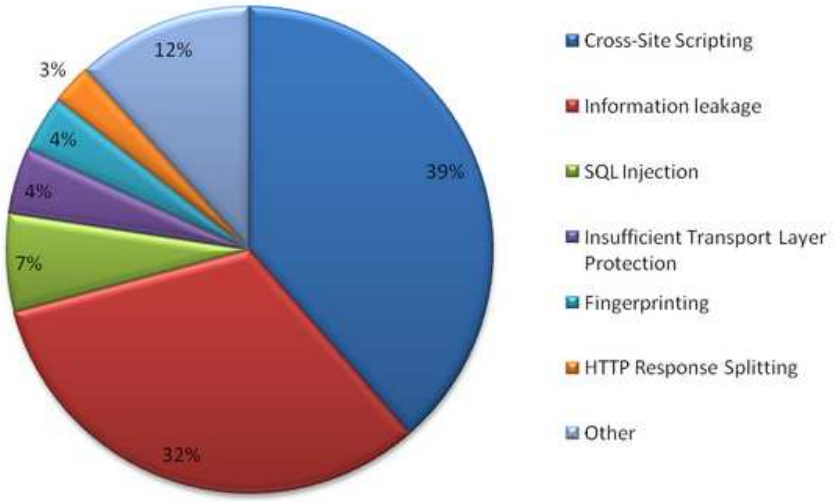


Figure 2.1: WASC Report: Percent of Vulnerabilities Statistical Analysis [5]

classified by kind of vulnerability. From the graph, we see that information leakage vulnerabilities ranks second, and is a high percentage of the total.

Thus, data security is an urgent challenge to many organizations [6]. Cryptography can be used in many cases to solve the problem of data security. The following sections will explore more about different encryption methods and facilities.

2.2 Cryptography

Cryptography is the practice and study of ”hiding” information (the Geek word origins mean ”hidden writing”), and the history of cryptography begins thousands of years ago. The first cipher-based encryption was used by Romans in the 1st century BC. The Caesar cipher, also known as the Caesar shift, is one of the simplest and most widely known encryption techniques [7]. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. The Caesar cipher was used for Roman military cryptography, and was still in use even as late as 1915. Today, it can still be found in children’s toys such as secret decoder rings.

Modern encryption begins in the 20th century during the first World War [8]. During the second World War, the invention of complex mechanical and electromechanical machines, such as the Enigma rotor machine, provided more sophisticated and efficient means of encryption. And the subsequent introduction of electronics and computing has allowed schemes of still greater complexity. Until the 1970s, secure cryptography was largely limited to govern-

ment agencies. However, the creation of a public encryption standard (DES), and the invention of public-key cryptography made encryption available to the public. Nowadays, with more attention to the data security, cryptography is developing very quickly. Different kinds of encryption facilities and algorithms appear frequently.

Cryptography helps protect data from being viewed or modified and helps provide a secure means of communication over otherwise insecure channels [9, 10, 11]. For example, data can be encrypted using a cryptographic algorithm, transmitted in an encrypted state, and later decrypted by the intended party. If a third party intercepts the encrypted data, it will be difficult to decipher.

Normally, cryptography can be used to implement the following goals:

- Confidentiality: To help protect a user's identity or data from being read.
- Integrity: To help protect data from being altered.
- Authentication: To assure that data originates from a particular party.

To achieve these goals, people can use some cryptographic primitives, known as the combination of algorithms and practices, to create a cryptographic scheme. Here is the list of the main cryptographic primitives:

1. Secret-key encryption (symmetric cryptography): Performs a transformation on data, keeping the data from being read by third parties. This type of encryption uses a single shared, secret key to both encrypt and decrypt data.
2. Public-key encryption (asymmetric cryptography): Performs a transformation on data, keeping the data from being read by third parties. This type of encryption uses a public/private key pair to encrypt and decrypt data. Data is encrypted by the public key and can only be decrypted by the private key, which is always kept secret.
3. Cryptographic hashes: Maps data of any length to a fixed-length byte sequence. Hashes are statistically unique; different two-byte sequences will not hash to the same value. Hashed data cannot be decrypted. This method is typically used to encrypt passwords.
4. Cryptographic signing: Helps verify that data originates from a specific party by creating a digital signature that is unique to that party. This process also uses hash functions.

Most encryption facilities today use Public-key encryption as the primary cryptographic primitive.

2.2.1 Cipher-Encryption Algorithms

The encryption algorithm is the mathematical procedure for performing encryption on data [12]. Through the use of an encryption algorithm, information is made into meaningless cipher text and requires the use of a key to transform the data back into its original form. There are many encryption algorithms being used today. The following is a general introduction to some common encryption algorithms.

RSA

RSA is a public-key cryptosystem for both encryption and authentication, and it was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman (the first letters of their names make up for "RSA") [13]. It is also the first algorithm known to be suitable for signing as well as encryption, and was one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations. It works as follows: take two large primes, p and q, and find their product n = pq (n is called the modulus). Choose a number, e, less than n and relatively prime to (p-1)(q-1). Find another number d such that (ed-1) is divisible by (p-1)(q-1). The values e and d are called the public and private exponents, respectively. The public key is the pair (n,e); the private key is (n,d). The factors p and q may be kept with the private key, or destroyed.

DES

DES stands for Data Encryption Standard, and it is a block cipher selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976 [14]. DES encrypts and decrypts data in 64-bit blocks, using a 64-bit key, although the effective key strength is only 56 bits. It takes a 64-bit block of plaintext as input and outputs a 64-bit block of ciphertext. Since it always operates on blocks of equal size and it uses both permutations and substitutions in the algorithm, so DES is both a block cipher and a product cipher. The actual encryption or decryption is performed by the main DES algorithm core function. DES has 16 rounds, meaning the core function algorithm is repeated 16 times to produce the ciphertext. It has been found that the number of rounds is exponentially proportional to the amount of time required to find the key using a brute-force attack. So as the number of rounds increases, the security of the algorithm increases exponentially. For DES, the same algorithm can be used for encryption as well as decryption.

Unfortunately, over time, various shortcut attacks were found that could significantly reduce the amount of time needed to find a DES key by brute force. As computers became progressively faster and more powerful, it was recognized that a 56-bit key was simply not large enough for high security applications. As a result of these serious flaws, NIST abandoned their offi-

cial endorsement of DES in 1997 and began work on a replacement, which is called the Advanced Encryption Standard (AES). Despite the growing concerns about its vulnerability, DES is still widely used by financial services and other industries worldwide to protect sensitive on-line applications.

IDEA

IDEA is short for the International Data Encryption Algorithm, which is a block cipher designed by James Massey of ETH Zurich and Xuejia Lai and was first described in 1991 [15, 16]. The algorithm was intended as a replacement for the Data Encryption Standard (DES). IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical transformations and an output transformation. The processes for encryption and decryption are similar. IDEA derives much of its security by interleaving operations from different groups' modular addition and multiplication and bitwise exclusive OR (XOR), which are said to algebraically "incompatible."

Blowfish

Blowfish is a symmetric block cipher designed to be used as a replacement for DES or IDEA [17, 18]. It was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm.

Blowfish is a symmetric chiper that uses a key length from 32 to 448 bits. It encrypts data in 64-bit block. Blowfish first convents the key into several subkey arrays. These subkeys are successively used and permuted during multiple rounds of addition and XOR-based encryption. In all, the Blowfish encryption algorithm will run 521 times to process all the subkeys and permutations.

Blowfish was one of the first secure block ciphers not subject to any patents and therefore freely available for anyone to use. This benefit has contributed to its popularity in cryptographic software.

Twofish

Twofish, first published in 1998, is a symmetric key block cipher algorithm [19]. It is a 128-bit block cipher that accepts a variable-length key up to 256 bits. The cipher is a 16-round Feistel network, a pseudo-Hadamard transform, bitwise rotations, and a carefully designed key schedule. The design of both the round function and the key schedule permits a wide variety of tradeoffs between speed, software size, key setup time, and memory. Depending on on the key length as well as whether Twofish is used for hardware based or software based encryption, Twofish may outperform AES in terms of speed.

CAST

Cast is encryption algorithm similar to Blowfish. It is a block cipher used in a number of products. It was designed by Carlisle Adams and Stafford Taveres, and name "CAST" represents the first letters of their names [20]. Cast-128 is licence-free algorithm available to everyone. It is a 12- or 16-round Feistel network with a 64-bit block size and a key size of between 40 to 128 bits (but only in 8-bit increments). Components include key-dependent rotations, modular addition and subtraction, and XOR operations. There are three alternating types of round function, but they are similar in structure and differ only in the choice of the exact operation at various points.

CAST-256 is a symmetric cipher designed in accordance with the CAST design procedure. It is an extension of the CAST-128 cipher and has been submitted as a candidate for NIST's Advanced Encryption Standard (AES) effort.

AES

AES is short for Advanced Encryption Standard, which is an encryption standard adopted by the U.S government [10, 21]. The standard comprises three block ciphers, AES-128, AES-192 and AES-256. Each AES cipher has a 128-bit block size, with key sizes of 128, 192 and 256 bits respectively.

Generally speaking, AES is an algorithm that starts with a random number, and then the key and data encrypted with it are scrambled through four rounds of mathematical processes. The key that is used to encrypt the number must also be used to decrypt it. The four rounds are called SubBytes, ShiftRows, MixColumns, and AddRoundKey. During SubBytes, a lookup table is used to determine what each byte is replaced with. The ShiftRows step has a certain number of rows where each row is shifted cyclically by a particular offset (while leaving the first row unchanged). The MixColumns step is a mixing operation using an invertible linear transformation in order to combine the four bytes in each column. The AddRoundKey derives round keys from Rijndaels key schedule, and combines each byte of the state with the corresponding byte from the round key. Lastly, these steps are repeated again for a fifth round, omitting the MixColumns step.

AES provides strong encryption and has been selected by NIST as a Federal Information Processing Standard in November 2001. In June 2003, the U.S. Government (NSA) announced that AES is secure enough to protect classified information up to the TOP SECRET level. Ultimately, anyone can use AES encryption methods, and it is free for public or private, commercial or non-commercial use. Most of encryption facilities that this thesis will study use AES as the default encryption algorithm.

2.2.2 Encryption Key Management

As we have seen, there are two components required to encrypt data: an algorithm and a key. The algorithm is generally known, and the key is kept secret. The key is a very large number that should be impossible to guess, and of a

size that makes exhaustive search impractical. In a symmetric cryptosystem, the same key is used for both encryption and decryption. In an asymmetric cryptosystem, the key used for decryption is different from the key used for encryption. It is worth noting that keys are different from passphrases. A passphrase is a password that provides an additional layer of protection for an encryption key.

There are many commercial technologies on the market today that are capable of transparently encrypting data at the file, database or storage-media level. Normally, these technologies have their own built-in proprietary key management methods. The following are some concepts introduction related to encryption key management [22, 23].

Trusted Platform Module (TPM)

The Trusted Platform Module (TPM) is a microcontroller security chip used to defend the internal data devices against real intelligent attacks. Normally, the TPM crypto processor is embedded in the computer motherboard [24]. Currently, TPM technology is used by the Microsoft Windows operating system. It is used to store encryption keys for and to authenticate a hardware device. Since each TPM chip is unique to a particular device and is capable of performing platform authentication, it can be used to verify that the system seeking the access is the expected system. Also, at boot time, the TPM chip can verify the integrity of key files before providing access to an encrypted disk. The way that TPM works with Bitlocker key management will be discussed in a later section.

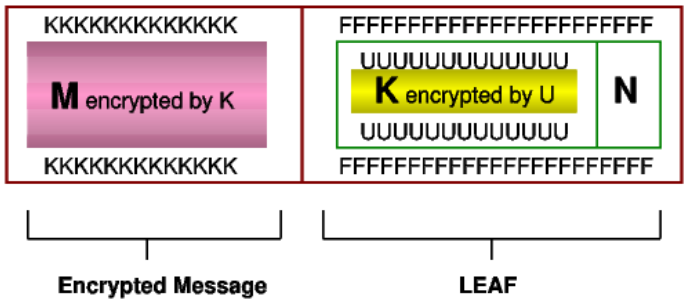
To allow for recovery in cases where the TPM module cannot release the key, additional copies of the key can be stored. The most common methods are to store a recovery key on a USB device or to create a recovery key consisting of a long numeric password. If the TPM interaction fails, the user will be asked to provide either the USB device containing the recovery key or the password.

Key Escrow

Key escrow is a storage arrangement for the keys needed to decrypt the encrypted data [25, 26]. It can be considered as an encryption system in which one or more ”escrow agents” hold copies of all encryption keys, so that, under certain circumstances, an authorized third party may gain access to those keys. These third parties may include businesses, who may want access to employees’ private communications, or governments, who may wish to be able to view the contents of encrypted communications.

In April 1993, President Clinton announced the Escrowed Encryption Initiative, ”a voluntary program to improve security and privacy of telephone communications while meeting the legitimate needs of law enforcement.” However, public response to the key escrow encryption was overwhelmingly negative. Despite the negative public opinion, on February 4, 1994, the Department of Commerce announced the approval of the Escrowed Encryption Standard as a voluntary Federal Information Processing Standard.

One implementation, the Clipper Chip breaks messages up into chunks, encrypts each chuck, and adds a Law Enforcement Access Field (LEAF), before transmitting the message. A simple structure of each encrypted and packaged chunk is as following:



- F = Family key (common to all Clipper Chips) - 80 bits
- N = serial Number of chip - 30 bits
- U = secret key for chip - 80 bits
- K = Key specific to particular conversation - 80 bits
- M = the Message

Nowadays, key escrow is still used in some systems to implement key recovery scheme. For example, Microsoft Exchange Server provides key escrow services for secure mail so you can recover encrypted data if private keys are lost or damaged.

LUKS

LUKS is short for Linux Unified Key Setup, and it is the standard for Linux hard disk encryption key management [27, 28]. It provides secure management of multiple passwords for a single device. By using the kernel device mapper subsystem, LUKS provides a low-level mapping that oversees encryption and decryption of the device's data (although actual encryption is performed by dm-crypt discussed below). LUKS stores all the necessary setup information in the partition header, using the TKS1, a template design developed for secure key setup on cross-platform. However, LUKS has some limitations. It is not well-suited for applications requiring more than eight users to have distinct access keys to the same device, nor to applications requiring file-level encryption.

TKS1

As mentioned above, TKS1 is a design template for secure key management [29]. It is an anti-forensic, two level, and iterated key setup scheme. Figure2.2 shows us the overall structure of TKS1. The salt and the AF-splitted master key come from key storage. The passphrase comes from an entropy-weak source like the user's keyboard input.

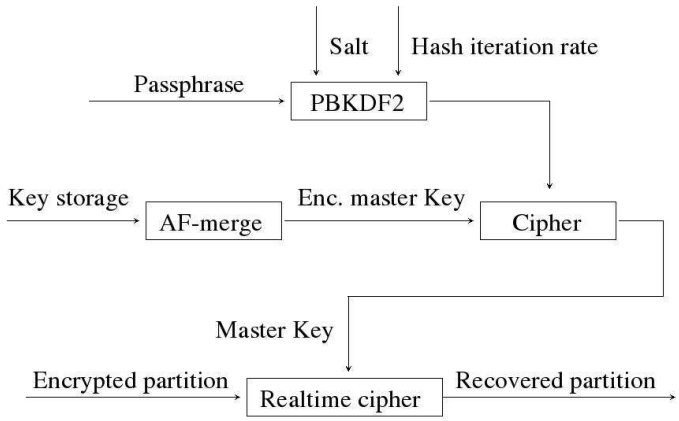


Figure 2.2: TSK1 Key Management Scheme

The initialization of the TSK1 system is straightforward: First, generate a master key and a salt to use for PBKDF2, and choose an appropriate hash iteration rate by benchmarking the system. Then have the user enter the passphrase, and process the passphrase with PBKDF2, thus obtaining the password-derived key. Thirdly, set up the master key cipher with the password derived key, and encrypt the master key with the master key cipher. After that, AF-split the encrypted master key, and save the AF-splitted encrypted master key, the iteration rate and the password salt to storage. Once the real time cipher is set up with the master key, master key copy is destroyed in memory.

The recovery of an encrypted volume happens as follows: First, read the salt, iteration rate and the AF-splitted encrypted master key from storage. Then AF-merge in memory, obtaining the encrypted master key. Then have the user enter the passphrase, and process the passphrase with PBKDF2, thus obtaining the password-derived key. Finally, set up the master key cipher with the password-derived key, and decrypt the encrypted master key with the master key cipher. Once again, the master key copy in memory is destroyed after use.

When a password has been compromised, the master key can be recovered as shown above, but instead of using it for the real time cipher, it can be re-encrypted using a new passphrase derived with PBKDF2. The master key encrypted with the old, compromised passphrase can be easily destroyed.

2.3 Encryption Strategies

File-based encryption was the first encryption strategy offered on modern computes. However, ordinary users are often not sophisticated enough to manually encrypt and decrypt files. For this reason, transparent encryption can be a more appropriate solution. Nowadays, filesystem level encryption and block device encryption are very popular; both of them are transparent encryption facilities. The following section presents each of these encryption strategies.

2.3.1 File-based Encryption

File-based encryption means encrypting the contents of a file, or part of the contents of a file [30, 31]. Although this method is sometimes referred to as folder encryption because all of the files in a folder can generally be encrypted with one action, the technology operates at the individual file level. This means that one can encrypt just those files that contain sensitive data and leave all other files unencrypted. Of course, files are decrypted or encrypted only for users who properly authenticate themselves by knowing the encryption key.

A file-based encryption solution allows users to apply encryption to just a few files until they gain confidence that either an operator error or technology problem won't destroy their data. However, it still has a number of disadvantages that need to be well understood. Firstly, file-level encryption can be very difficult to deploy and manage from a policy point of view. Organizations need to determine what data needs to be encrypted. Another disadvantage is that it depends on the user's action. Since users can inadvertently forget to encrypt a file that should be encrypted, or intentionally choose not to, the whole security system is very prone to human weaknesses. Thirdly, sometimes it is impossible, or at least impractical, to encrypt specific bits of sensitive data within an application. For example, there is no way in Microsoft Outlook to encrypt specific fields or a specific record within the Contacts database. The only option is to encrypt all Outlook database files, which can significantly degrade performance. Nevertheless, as the first generation of encrypted storage technology, file-based encryption was an important step in the development of encryption features.

2.3.2 Block Device Encryption

Block device encryption protects the data on a block device by encrypting it [32]. To access the device's decrypted contents, a user must provide a passphrase or key for authentication. This provides additional security beyond existing OS security mechanisms, which can protect the device's contents even if it has been physically removed from the system. With block device layer encryption, the user creates the filesystem on the block device, and the encryption layer transparently encrypts the data before writing it to the actual lower block device.

One advantage of block device layer encryption is that attackers learn nothing about the filesystem unless they have the key. For instance, attackers will not even know the type of filesystem or the directory structure. Also, sparse files can be securely and efficiently supported in filesystems on encrypted block devices.

However, block device encryption can have disadvantages that stem from the lack of integration with the filesystem itself: Firstly, a fixed region of storage must be pre-allocated for the entire filesystem. Resizing the partition later is often an inconvenient process. Secondly, it can be difficult to change encryption keys or ciphers, and there is no flexibility for the block device encryption mechanism to encrypt different files with different keys or ciphers (Dm-crypt is an exception when it works with LUKS). Thirdly, applications such as incremental backup utilities need access to the unencrypted data. Fourthly, data from the encrypted device is still sent in the clear over the network. Finally, all content in the filesystem incurs the overhead of encryption and decryption, including data that does not require secrecy.

Full Disk Encryption

Full disk encryption, can be considered as a special situation of block device encryption. It is also known as whole disk encryption or on-disk encryption, which is a kind of disk encryption software or hardware which encrypts every bit of data that goes onto a disk or disk volume [31, 32]. The term ”full disk encryption” is often used to signify that everything on a disk is encrypted, including special bootable operating system partitions.

The advantage of full disk encryption is that the disk is protected if it is physically removed from the computer. One drawback of full disk encryption is that it does not encrypt data during the process of transmission when the information is being shared between devices or stored on portable devices such as a flash drive or external hard drive. It also does not protect data that is being transferred via a network.

2.3.3 Filesystem-level Encryption

Filesystem-level encryption is a form of disk encryption where the entire filesystem contents are encrypted and decrypted by the operating system [31, 32]. The data actually on disk remains encrypted. In Linux systems, the filesystem is mounted via the loopback mount mechanism to provide an unencrypted view.. On Windows systems, each file has a unique encryption key.

The advantages of filesystem-level encryption include: Firstly, flexible file-based key management, so that each file can potentially be encrypted with a separate encryption key. Secondly, individual management of encrypted files: e.g., incremental backups can be made of the individual changed files even in encrypted form, rather than backup of the entire encrypted volume. Thirdly, access control can be enforced through the use of public-key cryptography, and the fact that cryptographic keys are only held in memory while the file that is decrypted by them is open.

However, unlike disk-based encryption, general-purpose file systems that include filesystem-level encryption do not typically encrypt file system metadata, such as the directory structure, file names, sizes or modification timestamps. This can be problematic if the metadata itself needs to be kept confidential. In other words, if files are stored with identifying file names, anyone who has access to the physical disk can know which documents are stored on the disk, although not the contents of the documents.

2.4 Encryption Facilities Introduction

There are many available encryption tools. This thesis will consider the following ones. Figure2.3 gives a general comparison about these facilities.

Tools	eCryptfs	Dm-crypt	TrueCrypt	Bitlocker	EFS
Running Platforms	Linux Kernel 2.6.19 or higher	Linux Kernel 2.6 or higher	Linux Kernel 2.6.5 or higher, Windows XP/Vista/7, Mac OS X	Window Vista/7 Ultimate and Enterprise, Window Server 2008	Windows 2000/XP/Vista/7, Windows Server 2003/2008
Encryption Strategy	Filesystem-level	Block Device	Block Device	Full Disk	Filesystem-level
Licensing	GPL	GPL	Free Open Source	Proprietary	Proprietary
TPM	Yes	No	No	Yes	No
Multiple Keys	Yes	with LUKS	No	Yes	No
Multiple Users	Yes	Yes	Yes	Yes	Yes

Figure 2.3: Overview of Different Encryption Facilities

2.4.1 eCryptfs

eCryptfs is an enterprise-class, kernel-native stacked cryptographic filesystem for Linux [33, 34]. By layering on top of the filesystem layer, eCryptfs protects files no matter the underlying filesystem type, partition type, etc.

eCryptfs uses the Linux kernel keyring service, the kernel cryptographic API, the Linux Pluggable Authentication Modules (PAM) framework, OpenSSL, the Trusted Platform Module (TPM), and the GnuPG keyring in order to make the process of key and authentication token management seamless to the end user. The architecture of eCryptfs can be seen in Figure 2.4.

The encryption algorithms used by eCryptfs can be: AES, Blowfish, DES3_EDE, Twofish, CAST6, and CAST5, and AES is most used today by users. eCryptfs encrypts and decrypts individual data extents in each file using a unique randomly generated File Encryption Key (FEK). The FEK is encrypted with the

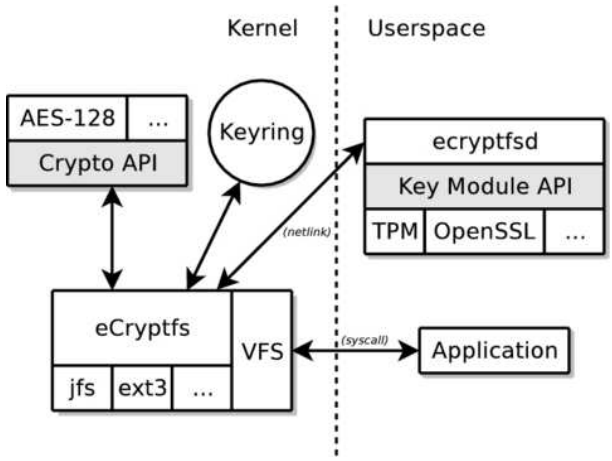


Figure 2.4: The Architecture of eCryptfs [33]

File Encryption Key Encryption Key (FEKEK), and the resulting Encrypted File Encryption Key (EFEK) is stored in the header of each file. Once the eCryptfs filesystem is loopback-mounted to the mount point successfully, files written to the mount point will be encrypted transparently and written to the directory in the underlying filesystem. To decrypt files that exist in the underlying filesystem, the eCryptfs volume needs to be mounted with the correct key.

2.4.2 TrueCrypt

TrueCrypt is a free open-source software application used for real-time encryption [36]. It is distributed without cost, and has source code available. It can create a virtual encrypted disk within a file or a device-hosted encrypted volume consisting of either an individual partition or an entire storage device. Also, TrueCrypt is compatible on different platforms, such as Windows XP/Vista/7, Mac OS X, and Linux.

TrueCrypt has no limitation to the size of file on encrypted disk volume. The size of virtual volume is only limited by the filesystem of the encrypted volume disk. i.e. on a FAT32 format disk, the size of file system created on the encrypted volume cannot be larger than 4 GB.

The TrueCrypt volume initialization process proceeds as following [37]:

1. Generate a series of keys based on entered password, using the key derivation function selected by the user.

2. Use these keys to encrypt the data in the volume; unused space within the volume is filled with random data at creation.

3. The encryption keys are stored within the volume header used by TrueCrypt (see Table 2.1), beginning at byte 256.

4. An additional encryption key is used to encrypt the volume header. It is generated from the password and a random salt using the selected key

$Offset$ ($bytes$)	Size (bytes)	Encryption Status	Description
0	64	Unencrypted	Salt
64	4	Encrypted	ASCII string ”TRUE”
68	2	Encrypted	Volume header format version
70	2	Encrypted	Minimum version required to open the volume
100	8	Encrypted	Size of volume
108	8	Encrypted	Byte offset of the start of the master key scope
116	8	Encrypted	Size of the encrypted area within the master key scope
252	4	Encrypted	CRC-32 checksum of the (decrypted) bytes 64-251
256	Var	Encrypted	Concatenated primary and secondary master keys used to encrypt the data
131072	Var	Encrypted	Data area (master key scope)
S-131072	65536	Encrypted /Unencrypted	Backup header (encrypted with a different header key derived using a different salt)

Table 2.1: TrueCrypt Volume Format Specification

derivation function. The salt is stored unencrypted as the first 64 bytes of the header.

5. A backup volume header encrypted with the derivation function a different salt is stored in the last 65536 bytes of the volume.

The mount process for a TrueCrypt volume proceeds like this:

1. The volume header is read into RAM. Salt is extracted (bytes 0-63)
2. After user enter the password, the volume header key is created using salt and the appropriate header key derivation function. The latter is determined by trial and error in the next step.
3. The header is decrypted using the header key. Bytes 64-68 are tested against the ASCII string ”TRUE” and the checksum for bytes 256-511 is computed and compared with the stored value (at byte 72). This process is repeated for each available derivation function until one succeeds.
4. When the checks in step 3 are passed successfully, the header has been decrypted and the volume can be mounted.
5. Decryption of data within the volume is performed as needed using the stored keys (the header bytes 256 and following).

2.4.3 Dm-crypt

Dm-crypt is a transparent disk encryption subsystem in Linux kernel versions 2.6 and later [35]. It is part of the device mapper infrastructure, and uses cryptographic routines from the kernel's Crypto API. Figure 2.5 shows the workflow of Dm-crypt.

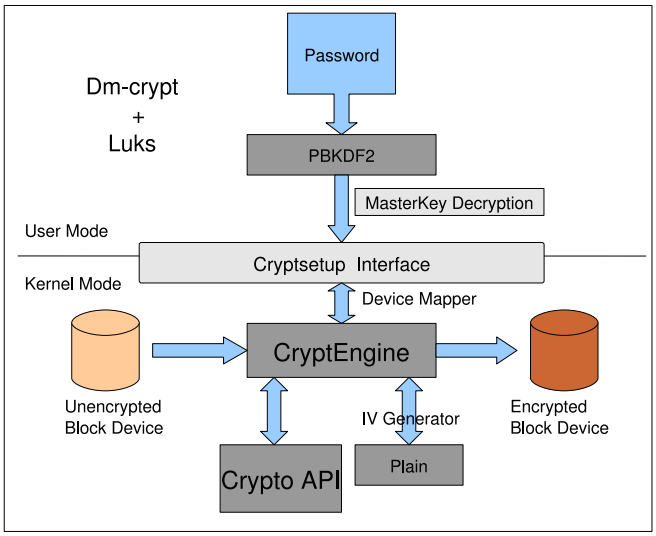


Figure 2.5: The Workflow of Dm-crypt Encryption [35]

Dm-crypt is implemented as a device mapper target and may be stacked on top of other device mapper transformations. Device mapper is a new infrastructure in the Linux 2.6 kernel, which can provides a generic way to create virtual layers of block device. Thus dm-crypt can encrypt whole disks, partitions, software RAID volumes and logical volumes. It appears as a block device, which can be used to hold a filesystem, a swap area or an LVM volume. Some Linux distributions support the use of dm-crypt for the root file system. These distributions use initrd to prompt the user to enter the passphrase at the console, or insert a smart card, prior to the normal boot process.

2.4.4 Bitlocker

BitLocker Drive Encryption is a full disk encryption feature included with Microsoft's Windows 7 Ultimate and Enterprise and Windows Server 2008 operating systems [39, 38]. It is designed to protect data by providing encryption for entire volumes. By default, it uses the AES encryption algorithm in CBC mode with a 128 bit key, combined with the Elephant diffuser for additional disk encryption-specific security (provided by AES). Bitlocker's architecture provides functionality and management mechanisms both in kernel mode and user mode. Figure 2.6 give an overview of the Bitlocker architecture [40].

BitLocker encryption has three implementation modes. Two modes require a TPM cryptographic hardware chip and a compatible BIOS, and they are Transparent operation mode and User authentication mode. The third mode does not have the TPM chip requirement, which is USB Key Mode. When the encryption process begins, a key is created. This key is called the Full Volume Encryption Key (FVEK), which is used to encrypt/decrypt the data. The FVEK is stored on the volume as part of the volume's metadata. To ensure the security of FVEK, it is encrypted by an additional key called the Volume

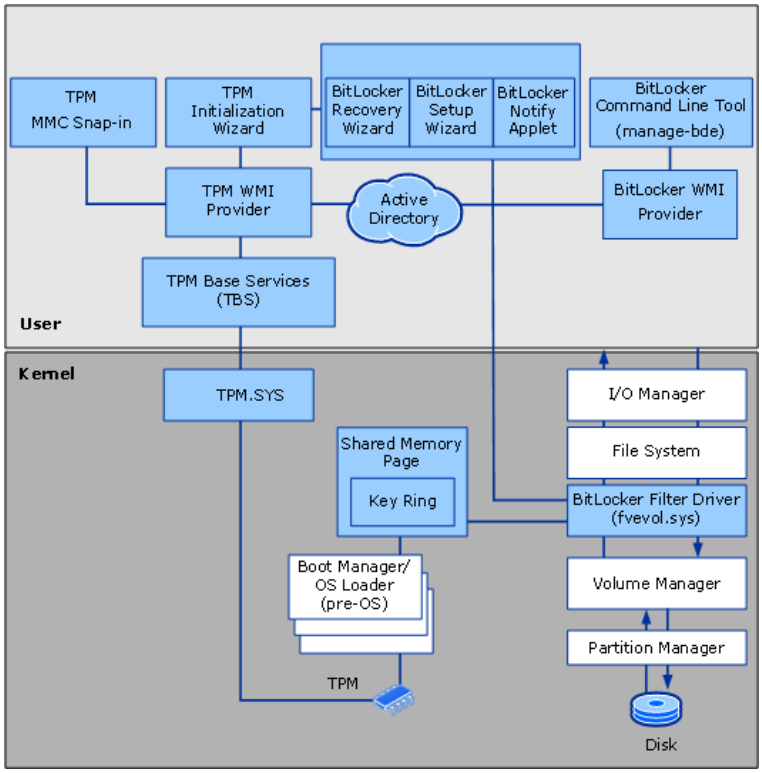


Figure 2.6: The Bitlocker Architecture [40]

Master Key (VMK). To encrypt a volume, Bitlocker must set the VMK first. To decrypt a volume, the OS boots, identifies the usage of Bitlocker, requests the VMK and uses it to access the FVEK, which in turn it provides access to the encrypted data. The workflow of Bitlocker is illustrated in Figure 2.7.

Windows 7 includes a new feature called "BitLocker To Go," which gives the lockdown treatment to portable storage devices like USB flash drives and external hard drives. On computers running Windows Vista or Windows XP, to open and view the content of removable drives that have been encrypted with BitLocker Drive Encryption in Windows 7, the Bitlocker To Go Reader (bitlockertogo.exe) program is need. This function makes Bitlocker more compatible on different Windows OS versions, so that people running Windows 7 are able to share their BitLocker-protected data on removable drives with anyone running Windows 7, Windows Vista, or Windows XP.

2.4.5 EFS

Encrypting File System (EFS) provides filesystem-level encryption for the Microsoft Windows environment [41]. The technology enables files to be transparently encrypted to protect confidential data from attackers with physical access to the computer. EFS is enabled in all versions of Windows meant for professional use from Windows 2000 onwards. However, since significant caveats exist for its use, no files are encrypted by default and EFS must be ex-

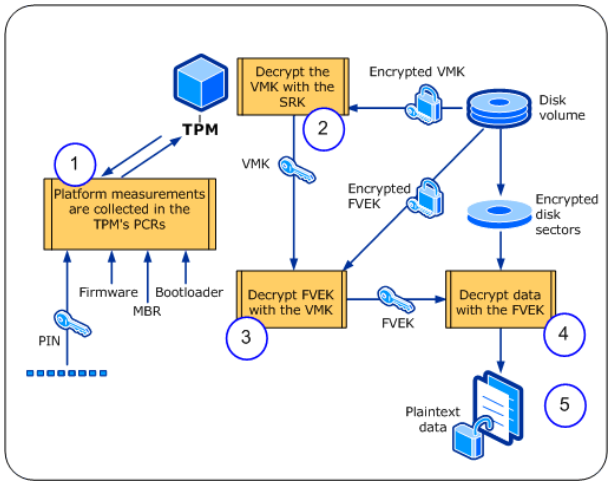


Figure 2.7: Bitlocker Encryption Workflow [40]

plicitly enabled by the user. More recent versions of EFS can also provide for sharing of encrypted files among multiple users.

There are several components in Windows system that make EFS work. Figure 2.8 is an overview about the EFS architecture [41]. EFS support is merged into the NTFS driver, and when NTFS encounters an encrypted file, NTFS executes EFS functions, which will encrypt and decrypt file data as applications access encrypted files. At the same time, CryptoAPI provides various cryptography services to applications. So all those components work together to make EFS function.

The process of using EFS to encrypt a file can be described as follows: First, EFS generates a File Encryption Key (FEK). Then, it employs a symmetric algorithm using the FEK to encrypt the file. The choice of algorithm depends on the version of the operating system. All versions of Windows 2000 use only the Expanded Data Encryption Standard (DESX). Windows XP Service Pack 1 or later and Windows Server 2003 support AES as well as DESX and 3DES, though AES is the default. Second, EFS extracts the public key from the EFS certificate contained in the user's profile. It encrypts the FEK with the user's public EFS key and stores it in the Data Decryption Field (DDF) in the header of the file. Also in the header of each encrypted file is a Data Recovery Field (DRF). The DRF contains an encrypted key created using the recovery certificate from each recovery agent. When a user opens an encrypted file, the user's private key decrypts the FEK in the DDF; then the FEK decrypts the file. Only the private key from the user who encrypted the file can decrypt the FEK. If necessary, a recovery agent can also decrypt the file using the encrypted FEK in the DRF. So only the user who encrypted the file and any designated recovery agents can access the file.

When two or more users require access to an encrypted file, an individual encrypted version of the FEK is created for each user, using their own public keys. These multiple encrypted FEKs are all stored in the file header's DDF.

Using EFS to encrypt a folder is much more efficient and safer than en-

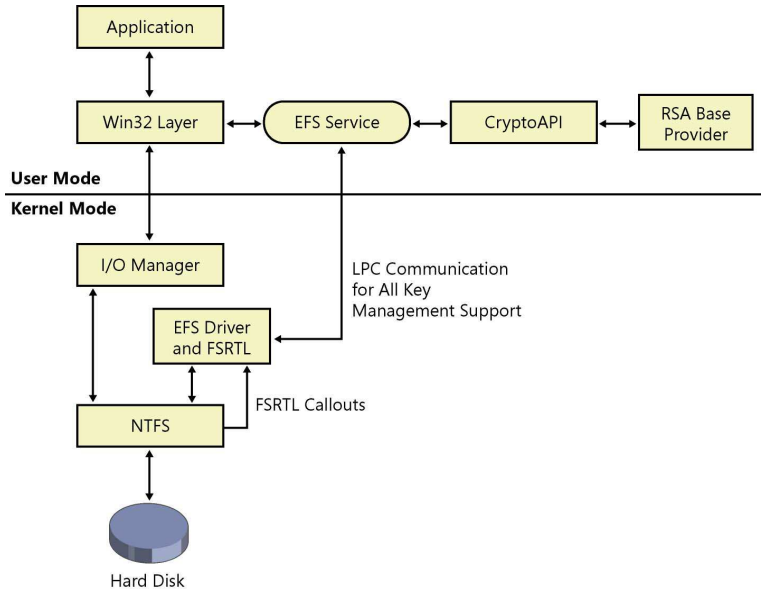


Figure 2.8: The Architecture of EFS [41]

crypting individual files. When a folder is encrypted, each file currently in the folder is encrypted and any file subsequently placed in the folder is also automatically encrypted. Each file in the folder is encrypted with a unique FEK. When encrypting an individual file, Windows makes a backup copy of the file, encrypts the original, and then deletes the backup. However, during this period, attackers could easily read the slack space using commonly available disk sector editing tools. While by marking a folder for encryption changes this process. All files stored in the folder are immediately encrypted; no backup versions ever exist on the drive, and this is much more safer.

Chapter 3

Methodology

This chapter covers the basic design of the experimental environment:

- Hardware equipments and software tools
- Experiment setup process on both Windows and Linux platforms
- Benchmark utilities used for performance analysis

3.1 Objective

As mentioned in the first chapter, currently there are many approaches to filesystem-level encryption and block device encryption. Ordinary users require a general overview of each approaches, in order to make choices based on their own requirements.

The project will install and use different encryption tools for filesystem-level encryption and block device encryption, and compare the performance of representative tasks with and without encryption for each encryption tool and platform. For the Windows operating system, tests will use Bitlocker, EFS, and TrueCrypt. For the Linux operating system, tests will use eCryptfs, Dm-crypt, and TrueCrypt. Based on the results of these experiments, the various solutions with respect to the dimensions such as effectiveness, ease of installation and use, and hard disk performance penalty will be discussed and evaluated. The following will cover the hardware and software used, and also give a short introduction to the set up process of each encryption facilities.

3.2 Tools and Equipment

3.2.1 Hardware Specification

The experiment environment uses a dual-boot computer. Table 3.1 shows the equipment and software specifications.

The following external storage devices were used for testing:

- 500 GB Seagate FreeAgent™ Go USB Hard Disk

- 8 GB Kingston DT101 II USB Stick

OperatingSystem	Windows	Linux
Version	Windows 7 Ultimate	Red Hat Enterprise Linux 5.4 Professional (2.6.18 kernel)
CPUType	Intel(R) Core(TM)2 CPU	Intel(R) Core(TM)2 CPU
Memory	2.00 GB	2.00 GB
HardDisk	WDC WD800adfs-75SLR2 ATA Device	ST380815AS ATA Device
HardDiskSize	80G	80G

Table 3.1: Hardware Specification

Tests	Sequential Access					Random Access				
	I/O Operation Size			Multi File	Directory Hierarchy	Seek Rate	I/O Operation Size			Multi File
	Small	Medium	Large				Small	Medium	Large	
Sequential I/O Tests		Y	Y	Y	Y					
Bonnie++	Y	Y		Y	Y	Y	Y	Y		Y
IOzone	Y					Y	Y			
Seekwatcher			Y	Y	Y					
HPC Calculation									Y	

Figure 3.1: Experiment Overview

3.2.2 Software Tests List

Here is a general list of the tests and their associated software applications used in the thesis. More details will be given in later sections.

- Sequential I/O tests were performed in all environments using standard operating system functionality: reading a 3 GB binary file, writing a 3 GB binary file, reading a 2.3 GB directory tree and writing a 2.3 GB directory tree.
- Bonnie++-1.03e [42]: an I/O benchmark available for Linux only.
- IOzone version 3.338 [43]: an I/O benchmark available for Windows and Linux.
- Seekwatcher-0.12 [44]: a low-level I/O analysis tool for Linux.

- Gaussian 09 computational chemistry software [45]: available for Windows and Linux and used to test large random I/O operations performance.
- Windows 2003 Server Resource Kit tools - timeit.exe [46]: used for CPU and elapsed time measurements.
- Linux time utility : used for CPU and elapsed time measurements.

Figure 3.1 summarizes all the tests in the experiment along with the types of I/O operations they exercise. With each encryption tool setup, ordinary user sequential I/O operations, I/O benchmarks and HPC calculations are executed as available.

3.2.3 Bonnie++ Benchmark

Bonnie++ is a benchmark suit that focuses on fundamental hard drive and file system performance [42]. As most applications that perform heavy I/O will not read or write data in single characters, the -f option was used to eliminate the suite's characted-level tests.

One example of the Bonnie++ command line and raw results can be seen as follows:

```
# bonnie++ -u root -d /mnt/crypt/ -f
Using uid:0, gid:0.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...
Create files in sequential order...done.
Stat files in sequential order...done.
Delete files in sequential order...done.
Create files in random order...done.
Stat files in random order...done.
Delete files in random order...done.
Version 1.03e ——Sequential Output—— –Sequential Input- –Random-
-Per Chr- –Block– -Rewrite- -Per Chr- –Block– –Seeks–
Machine Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP /sec %CP
pc116-74.iu.h 3536M 51060 12 23922 1 57163 0 163.7 0
——Sequential Create—— ———Random Create———
-Create– –Read— -Delete– -Create– –Read— -Delete–
files /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
16 16158 97 +++++ +++ +++++ +++ 16006 95 +++++ +++ +++++ +++
pc116-74.iu.hio.no,3536M,,,51060,12,23922,1,,,57163,0,163.7,0,16,16158,97,
+++++,+++,+++++,+++,16006,95,+++++,+++,+++++,+++
```

Bonnie++ benchmarks three things: data read and write speeds, number of seeks that can be performed per second, and number of file metadata operations that can be performed per second. Metadata operations include file creation and deletion as well as getting metadata such as the file size or owner.

Bonnie++ also reports (%CP column above) the percentage of the CPU that was used to perform the I/O for each test. The file metadata tests are shown in the second row of results; in them, files with a zero byte size are created, read, and finally deleted. The create, read, delete metadata tests are performed using file names that are sorted numerically and those are just random numbers. Some filesystems perform much better if an application creates and accesses files in a specific order. Because Bonnie++ performs the metadata tests twice, you can see whether a filesystem has optimized accesses to files by performing accesses in sorted file name order. Some of the metadata benchmarks are reported by Bonnie++ as +++++ instead of a real number per second. This happens when that particular benchmark completes too quickly.

3.2.4 IOzone Benchmark

IOzone is a free filesystem benchmark utility, and it is useful for performing broad filesystem analysis on Windows and Linux systems [43]. It can test file I/O performance for many different kinds of I/O operations.

In the experiment, IOzone 3.3888 version is executed on both Windows 7 Ultimate and Red Hat 5.4 platforms under different encryption facilities environments. The following operations were included in this study:

- Reader: sequential I/O read operations using a range of record and file sizes.
- Writer: sequential I/O write operations using a range of record and file sizes.
- Rereader: sequential I/O read operations using a range of record and file sizes from previously-accessed (cached) data.
- Rewriter: sequential I/O write operations using a range of record and file sizes to an existing file (avoiding metadata write).
- Random read: random access I/O read operations using a range of record and file sizes.
- Random write: random access I/O write operations using a range of record and file sizes.
- Record rewrite: Repeated writes of a record to the same disk location.

The IOzone command used was of the form:

#iozone -ae -q 1024 -y 32 -i 0 -i 1 -i 2 -i 4 -g 1G -n 64k -p -R -f *output-file -U* filesystem

where -n and -g specify the range of file sizes and -y and -q specify the range of record sizes in KB. -p purges the processor cache before each test, and -e includes buffer flushing times in the reported results. The remaining options specify the tests to run and output location and format. IOzone reports results in a tabular format starting with the smallest files up to the largest files. For each file size, the record size range is tested from smallest to largest record size.

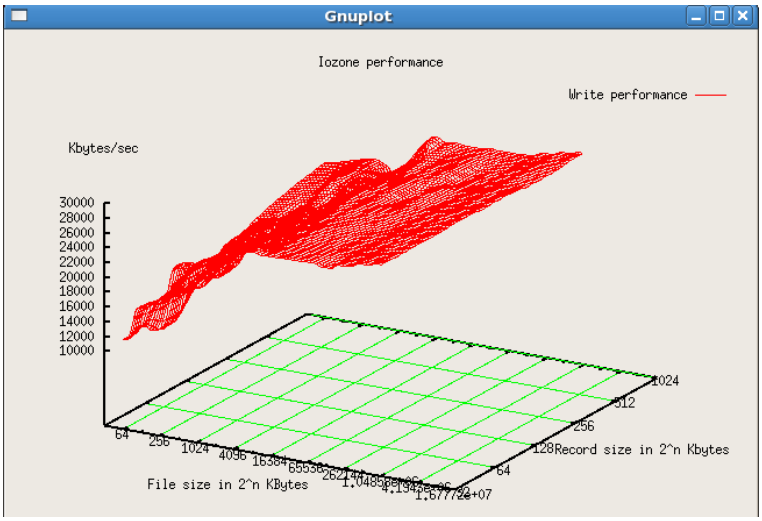


Figure 3.2: IOzone File System Performance Graph Plotted by Gnuplot

The IOzone package comes with scripts called Generate_Graphs and gengnuplot.sh to create graphs using gnuplot from its tabular output. Generate_Graphs calls gengnuplot.sh multiple times to generate a graph for each test that IOzone performs in its benchmark and then runs gnuplot to show each of these graphs and generates PostScript output at the same time. Generate_Graphs uses the gnu3d.dem file to drive the gnuplot operations. Figure 3.2 shows a 3D graph plotted by IOzone. While these plots are very attractive, this study will present these results in another form better suited to performance analysis.

3.2.5 Seekwatcher I/O

Seekwatcher is a Linux utility to visualize I/O patterns and performance by generating graphs from blktrace runs [44]. And there are three basic ways to run Seekwatcher:

- It can generate graphs from an existing blktrace run, start blktrace and run a program, or make a series of pngs from a single trace.
- It can run blktrace on multiple devices at the same time, and the resulting graph will have I/O from each device combined.
- It can make an animation of the IO generated by a given run.

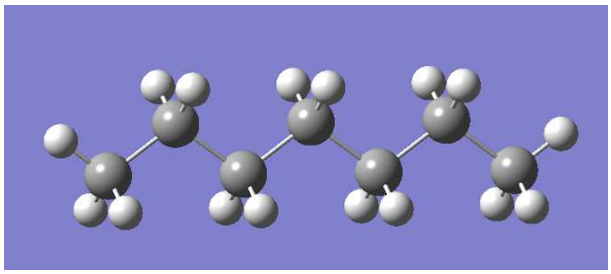
In this thesis, all tests related to Seekwatcher are running in a single trace by starting blktrace, executing sequential I/O operations and generating graphs. Seekwatcher uses matplotlib to generate graphs.

Each generated graph presents three subfigures graphing the Disk I/O Offset, Seek Count and Troughput through the course of the operation. For example, to test the I/O patterns and performance of hard disk partition when reading a large file, the command line can be as following form:

seekwatcher -t *trace-file -p* 'read command' -o *output-image -d* device

3.2.6 HPC Calculations - Gaussian 09

In order to test large I/O operations in random mode, a scenario not covered by often tests, the computational chemistry package Gaussian 09 was run for a problem that requires substantial I/O operations. The calculation performed a single point energy calculation for hexane (C_6H_{14}) using the highly accurate but computationally expensive coupled cluster method, including single and double excitations [47, 48]. A full integral transformation is performed to maximize the I/O requirements. The calculation uses the 6-31G(d,p) basis set, resulting in 170 basis functions for the molecule's 20 atoms. The calculation was set up so that all intermediate files were placed on the volume being tested.



3.3 Experiment Setup

Both Windows and Linux environments are considered in the thesis experiment, and different encryption facilities are tested. This section introduces the procedures for each experiment at setup in detail.

3.3.1 Windows OS Encryption Facilities Comparison

Bitlocker, TrueCrypt, EFS were tested on the Windows 7 platform for the sequential I/O operations, I/O benchmarks and HPC calculation. Performance results for each encrypted environment were compared with the unencrypted device data. Experiments related to Windows platform have three different hard disk environments options:

- Internal Hard Disk Partition (ST380815AS ATA Device, 48G)
- External USB Hard Disk Partition (Seagate FreeAgent™ Go, 50G)
- External USB stick (8G Kingston DT101 II)

Bitlocker Setup

On the Windows 7 Ultimate operating system, Bitlocker helps keep all the data safer by encrypting the entire drive. Once Bitlocker is turned on, any file you save on that drive is encrypted automatically.

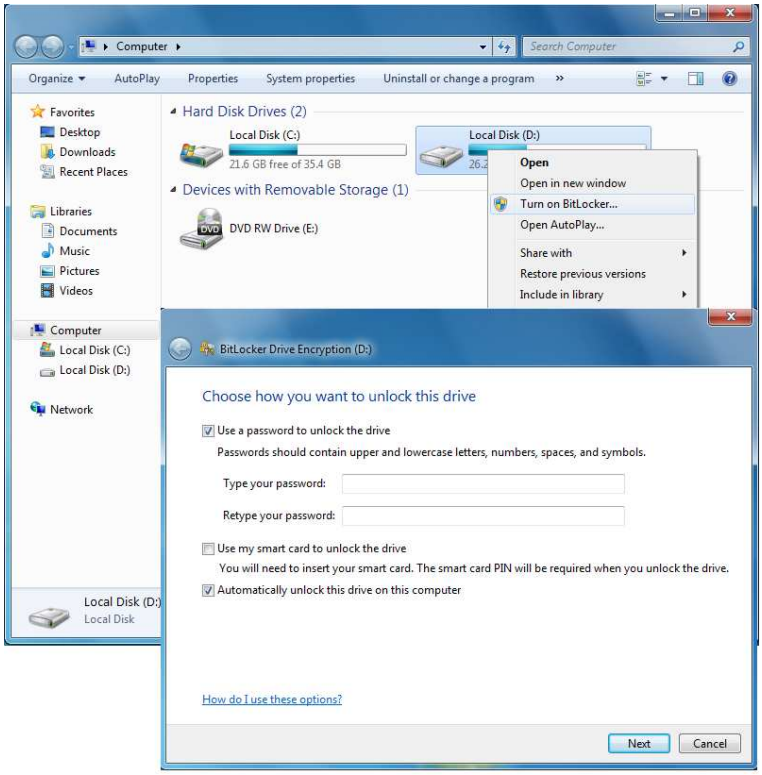


Figure 3.3: Bitlocker Encryption Setup

Figure 3.3 illustrates enabling Bitlocker to encrypt a hard disk drive. The operation of setting up Bitlocker is quite simple: Right click the hard disk icon, select "Turn on Bitlocker", specify a password and finally save the key file. After that, Bitlocker starts encrypting the device, a lock appears on that hard disk icon when it finishes. All the file on the virtual volume will be encrypted on the designated encrypted disk volume, enabling transparent access.

As mentioned in Chapter 2, Bitlocker encryption has three implementation modes. In this thesis, Bitlocker experiments are based on the USB key mode without TPM chip support. The Windows 7 Ultimate platform also has the new feature of Bitlocker to go to encrypt a removable USB device, and the setup process is almost the same as hard disk encryption above.

TrueCrypt Setup On Windows 7

TrueCrypt is free open-source disk encryption software available on multiple platforms. On Window 7 Ultimate, one must download and install the latest version TrueCrypt 6.3a setup.exe program from the official website. When starting the TrueCrypt program, the interactive GUI will display.

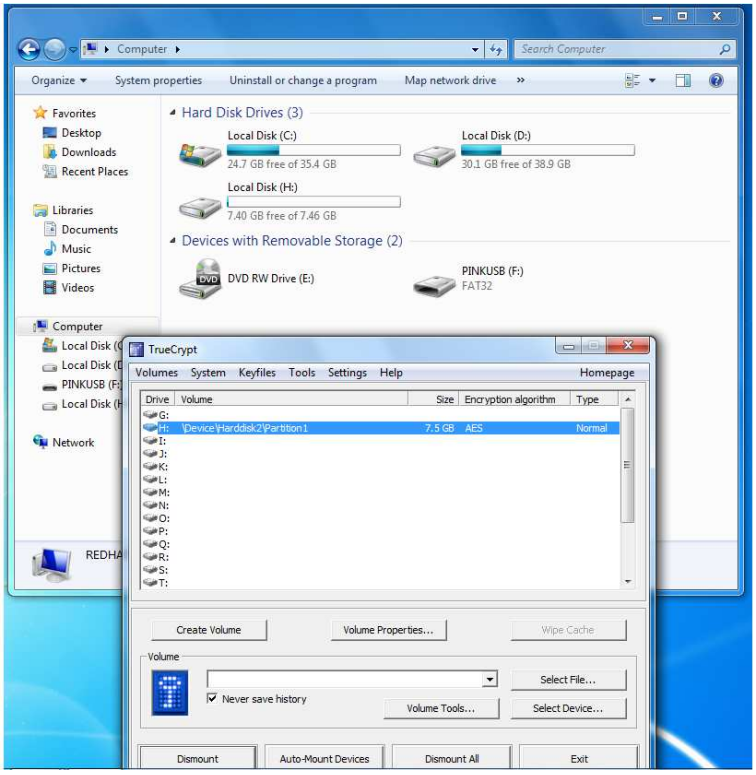


Figure 3.4: TrueCrypt Encryption Setup

The procedure for using TrueCrypt to encrypt volume is very simple. First, click create volume and select the device. Then, set password or keyfile, choose encryption algorithm and start volume encryption. After that, automount the encrypted volume to an unused virtual volume. In Figure 3.4, disk F is the encrypted volume, and disk H is the mounted virtual volume. Thereafter all the files copied on the virtual volume H will be encrypted on the designated encrypted disk volume F . Users who want to access the files on the virtual volume will be prompted for the password.

EFS Setup

EFS technology is supported by most Windows versions. It enables files to be transparently encrypted to protect confidential data from attackers with physical access to the computer.

The setup of EFS encryption is very straightforward. As seen in Figure 3.5, first the folder's Properties dialogbox is opened. Next, click Advanced, and

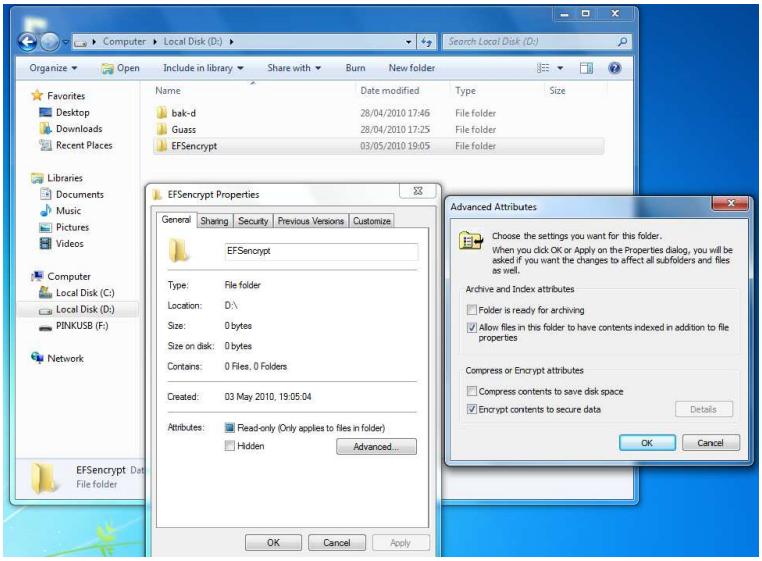


Figure 3.5: EFS Encryption Setup

then select the option ”Encrypt contents to secure data.” After applying this attribute, the folder icon will become green, which means EFS encryption is successful, and any files or subfolders under this folder will be automatically encrypted.

3.3.2 Linux OS Encryption Facilities Comparison

eCryptfs, Dm-crypt, TrueCrypt were tested on the Red Hat 5.4 platform for the sequential I/O operations, I/O benchmarks and HPC calculation. Performance results for each encrypted environment were compared with the unencrypted device data. Experiments related to Linux platform have three different hard disk environments:

- Internal Hard Disk Partition (ST380815AS ATA Device, 48G)
- External USB Hard Disk Partition (Seagate FreeAgent™ Go, 50G)
- External USB stick (8G Kingston DT101 II)

eCryptfs Setup

eCryptfs works by remounting a filesystem via the loop back mount mechanism. Files in the ”lower” filesystem are encrypted, and eCryptfs provides transparent encryption and decryption as needed. eCryptfs encryption on block device /dev/sdb3 can be set up as follows:

```
# mount /dev/sdb3 /mnt/data/
# mount -t ecryptfs /mnt/data/ /mnt/ecrypt/
Select key type to use for newly created files:
1) openssl
2) tspi
3) passphrase
Selection: passphrase
Passphrase:
Select cipher:
1) aes: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
2) blowfish: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
3) des3_ede: blocksize = 8; min keysize = 24; max keysize = 24 (not loaded)
4) twofish: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
5) cast6: blocksize = 16; min keysize = 16; max keysize = 32 (not loaded)
6) cast5: blocksize = 8; min keysize = 5; max keysize = 16 (not loaded)
Selection [aes] : aes
Select key bytes:
1) 16
2) 32
3) 24
Selection [16]: 32
Enable plaintext passthrough (y/n) [n] : n
Attempting to mount with the following options:
ecryptfs_unlink_sigs
ecryptfs_key_bytes=32
ecryptfs_cipher=aes
ecryptfs_sig=3be5d1f6400ba896
Mounted eCryptfs
```

The passphrase must be entered whenever the filesystem is mounted.

Dm-crypt Setup

On the Red Hat 5.4 platform with kernel 2.6.18, Dm-crypt uses a new infrastructure feature called Device mapper, which can provide a generic way to create virtual layers for a block device. The commands of setup Dm-crypt encryption with LUKS on block device /dev/sdb3 is as follows:

```
# cryptsetup luksFormat /dev/sdb3
WARNING!
========
This will overwrite data on /dev/sdb3 irrevocably.
Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
Command successful.
# cryptsetup luksOpen /dev/sdb3 CRYPTO
Enter LUKS passphrase for /dev/sdb3:
key slot 0 unlocked.
Command successful.

# mkfs.ext3 /dev/mapper/CRYPTO
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
7848960 inodes, 15697384 blocks
784869 blocks (5.00First data block=0
Maximum filesystem blocks=0
480 block groups
32768 blocks per group, 32768 fragments per group
16352 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 36 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

# mount /dev/mapper/CRYPTO /mnt/crypt/

# umount /mnt/crypt/
# cryptsetup luksClose CRYPTO
```

See Appendix A.1 for Dm-crypt setup without LUKS.

TrueCrypt Setup On Linux

TrueCrypt is a multi-platform tool for encryption. Its setup process on Windows 7 Ultimate has been introduced before. Red Hat 5.4 with kernel 2.6.18 can not initialize any Truecrypt-6.3a file systems due to an acknowledged bug

in their specific kernel build. Fixing the bug requires upgrading the Red Hat customized kernel to version 2.6.24 or later, a daunting task even for an experienced system administrator. Therefore, In the test, the Linux platform TrueCrypt encryption is implemented from GUI interface on Ubuntu 8.10 with kernel 2.6.24-27-generic on the same hardware. The command for setting up Truecrypt encryption is as follows:

 # truecrypt -t -c /dev/sdb3
Volume type:
1) Normal:
2) Hidden:
Select [1]: *1*
Encryption algorithm:
1) AES
2) Serpent
3) Twofish
4) AES-Twofish
5) AES-Twofish-Serpent
6) Serpent-AES
7) Serpent-Twofish-AES
8) Twofish-Serpent
Select [1]: *1*
Hash algorithm:
1) RIPEMD-160
2) SHA-512
3) Whirlpool
Select [1]: *1*

Filesystem:
1) FAT
2) None
Select [1]: *1*
Enter password:
Re-enter password:
Enter keyfile path [none]:
Please type at least 320 randomly chosen characters and then press Enter:
Done: 100.000The TrueCrypt volume has been successfully created.

 # truecrypt -t -k "" - -protect-hidden=no - -mount /dev/sdb3 /mnt/truecrypt/
Enter password for /dev/sdb3:

 # truecrypt -t -l
1: */dev/sdb3 /dev/loop0 /mnt/truecrypt*

Chapter 4

Measurements and Results

This chapter covers the experiment output and the final results. The following information is presented:

- Approximate Encryption Setup Time
- Sequential I/O Performance Comparison
- Bonnie++ Benchmark Results
- IOzone Benchmark Performance Reports
- Seekwatcher I/O Analysis
- HPC Calculation I/O Performance Comparison

The following sections present the data primarily in graphical form. See the Appendixes for raw data.

4.1 Approximate Encryption Setup Time

Figure 4.1 shows the approximate time required to set up encryption using the different encryption facilities. For eCryptfs, Dm-crypt, TrueCrypt and Bitlocker, the time is based on encrypting an empty 8G Kingston DT101 II USB stick. For EFS, the time is for encrypting an 8G folder with many files and subdirectories inside. EFS can be applied to an empty folder essentially instantaneously.

As we can see from the histogram, eCryptfs and Dm-crypt encryption take much less time than the other tools, EFS and Bitlocker are more time-consuming, and the time for TrueCrypt is in between. In addition, for the same size folder, EFS encryption setup time is affected by the number of subdirectories and files in the folder.

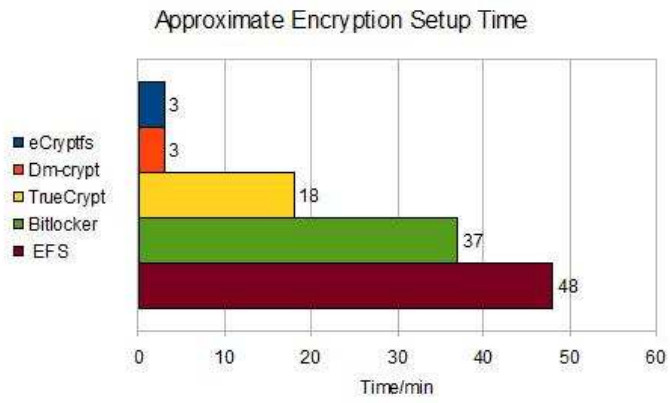
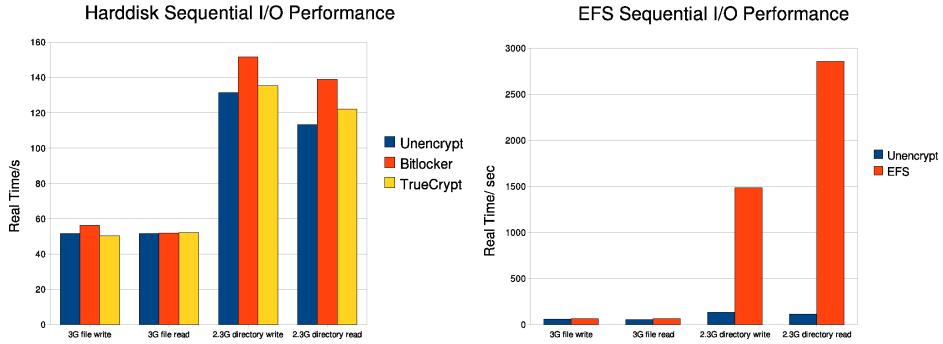


Figure 4.1: Encryption Facilities Approximate Setup Time Comparison

4.2 Sequential I/O Performance: Windows

Figure 4.2 displays the sequential I/O performance of different Windows encryption facilities. When executing sequential I/O operations for the 3G file and 2.3G directory, the Windows Resource Kit Utility timeit.exe is used to record the elapsed time.



(a) Bitlocker, TrueCrypt (b) EFS

Figure 4.2: Windows Encryption Facilities Sequential I/O Performance

Figure 4.2 presents the real time for sequential I/O operations based on Bitlocker, TrueCrypt encryption and the unencrypted state, which shows Bitlocker encryption takes a bit more real time than TrueCrypt. TrueCrypt encryption does not have much performance penalty for these sequential I/O operations.

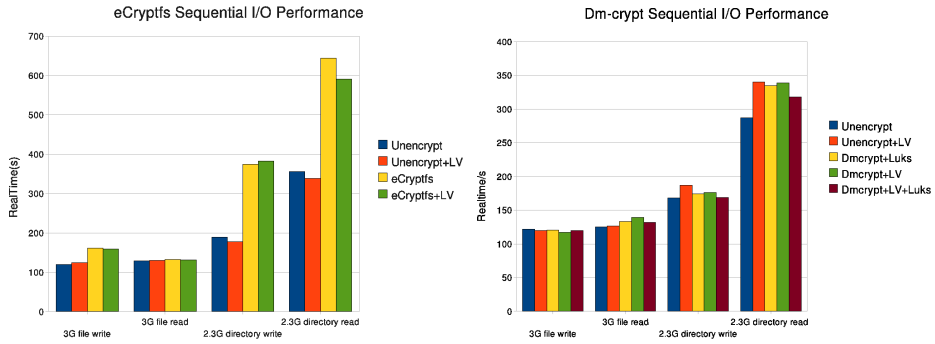
Figure 4.2(b) compares the real time difference for sequential I/O operations between EFS encryption and unencrypted state. For EFS encryption, directory sequential I/O operation takes much more time than without encryption.

4.3 Sequential I/O Performance: Linux

4.3.1 Logical Volume and LUKS Effects

Figure 4.3 shows the real time for performing sequential I/O on eCryptfs encrypted hard disk and Dm-crypt encrypted hard disk, with and without using logical volumes.

Figure 4.3(a) compares eCryptfs encryption with the unencrypted state, and also compares about the situations with and without logical volumes. The results show that the logical volume state does not affect sequential I/O performance very much. However, eCryptfs encryption doubles the real time of directory sequential I/O performance compared with the unencrypted state.



(a) eCryptfs (b) Dm-crypt

Figure 4.3: Logical Volume and LUKS Effect Analysis

Dm-crypt encryption, both with/without logical volumes and with/without LUKS are compared in Figure 4.3(b). The graph indicates that Dm-crypt encryption does not affect sequential I/O performance compared to the unencrypted device. In addition, whether logical volumes or LUKS are used has almost no effect on the sequential I/O performance.

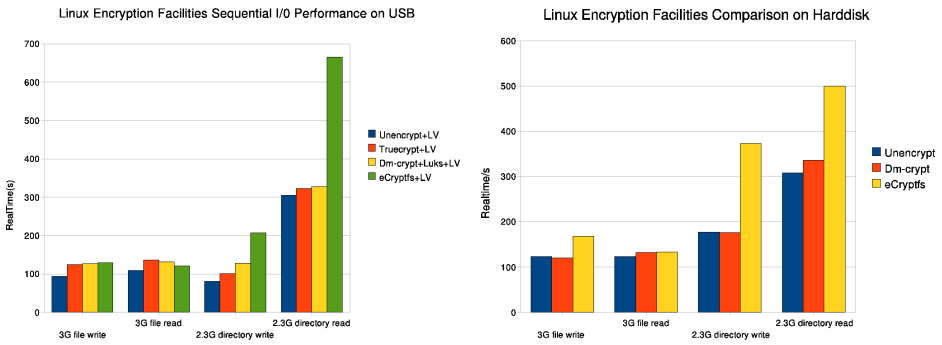
Based on these results, most subsequent tests use eCryptfs without a logical volume and Dm-crypt with LUKS but without a logical volume.

4.3.2 Internal Hard Disk and External Hard Disk Comparison

In Figure 4.4, there are two subfigures comparing Linux encryption facilities for sequential I/O performance using an external USB hard disk and the internal hard disk.

Figure 4.4(a) displays the real time difference of Linux encryption tools on a 50 GB external USB partition. In this case, the performance of Dm-crypt and TrueCrypt are quite close to the unencrypted situation. However, eCryptfs takes much more real time for directory sequential I/O operations.

The internal hard disk performance with Dm-crypt and eCryptfs is illustrated by Figure 4.4(b). Again, the sequential I/O performance for Dm-crypt



(a) External USB Hard Disk (b) Internal Hard Disk

Figure 4.4: Linux Encryption Facilities Sequential I/O Performance

encryption is not very different from unencrypted situation. In contrast, eCryptfs takes much more real time for directory write and read operations.

Thus, the relative performances of the various encryption facilities with respect to the unencrypted device does not change much with the slower I/O of the external USB disk.

4.3.3 TrueCrypt Encryption Comparison on Different Platforms

As mentioned in TrueCrypt setup section, Red Hat 5.4 with kernel 2.6.18 can not successfully create Truecrypt-6.3a volumes. So instead of using the Red Hat platform, TrueCrypt was initialized on an 8G USB stick via Ubuntu 8.10 platform with kernel 2.6.24-27-generic. This device was then able to be used on a Red Hat system. Figure 4.5 shows a comparison for TrueCrypt encryption sequential I/O performance on Windows 7 Ultimate and Red Hat 5.4 professional platforms.

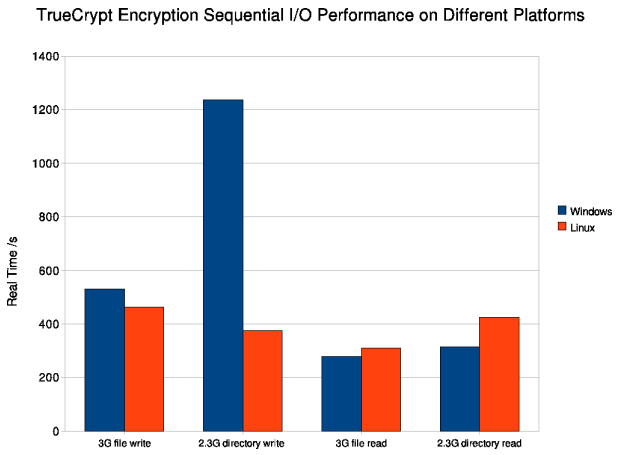


Figure 4.5: TrueCrypt Encryption Sequential I/O Performance Comparison

On this hardware, TrueCrypt encryption on the Windows platform takes more time for write operations than the Linux platform, most notably for the

directory write. However, the read operation results are opposite, although the differences are quite small.

The very poor performance of TrueCrypt for the directory write test is consistent with the Windows unencrypted results, where their operation takes almost 9 times as long as the corresponding read operation (2734 vs 306 seconds). The ratio for the large file operations is 1.6 (457 vs 288 seconds).

4.3.4 Different Tools Encryption on The Same Internal Hard Disk

3G file and 2.3 G directory sequential I/O operations were performed on the same internal hard disk partition in order to comapare I/O performance and tools across operating systems.

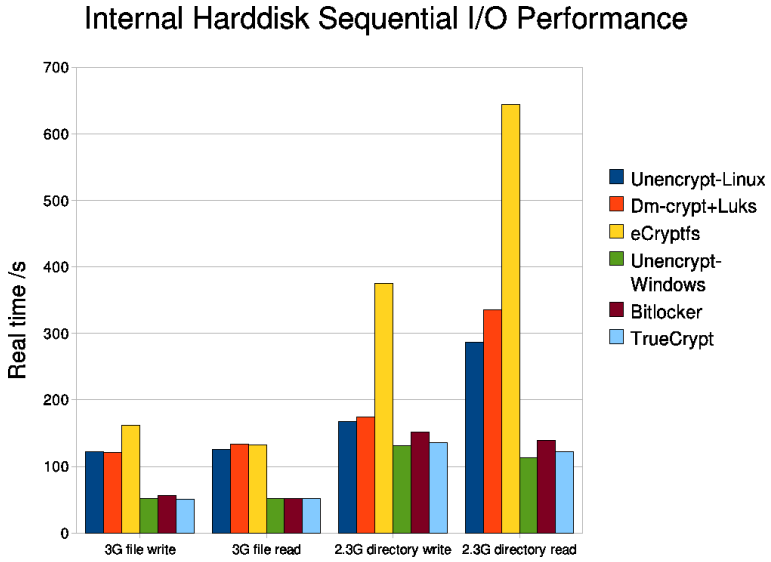


Figure 4.6: Different Tools Sequential I/O Performance Comparison

Figure 4.6 shows us that the same sequential I/O operation tests take more time on the Linux platform than on the Windows platform. For the Linux encryption tools, eCryptfs encryption takes the most time, and Dm-crypt encryption performs similar to the unencrypted state. For the Windows encryption facilities, Bitlocker encryption takes a bit more time than TrueCrypt encryption, but neither of them have much performance penalty compared with the unencrypted state.

These results also show that Linux I/O performance for these sequential I/O operations is much slower than under Windows. The unencrypted results measure this base line I/O level (see the first and fourth bars in each group in Figure 4.6). The performance gap between Linux and Windows is largest for the file-based operations (about a factor of 2) and shrinks to about 20% slower for the directory operations. The block device encryption schemes follow the same underlying I/O profile.

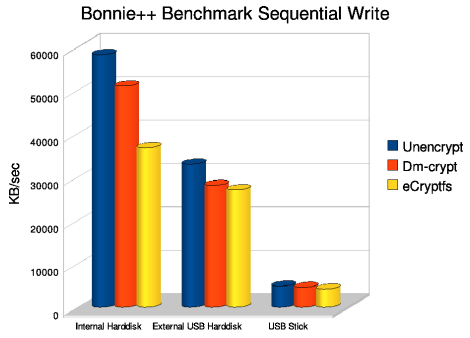
4.4 Bonnie++ Benchmark Results

In the experiment, we test Bonnie++ benchmark for different Linux encryption tools (unencrypted, Dm-crypt, eCryptfs) in three different hardware situations: internal hard disk, external USB hard disk, and USB stick.

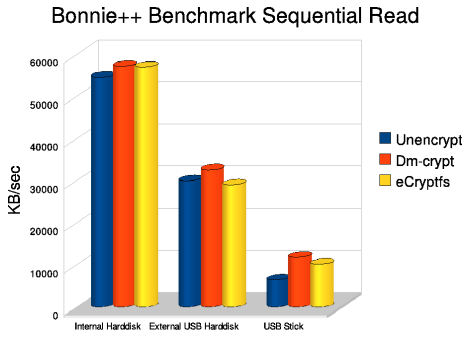
To make comparisons among the various Linux tools and different hardware situations, results are categorized into three sections below.

4.4.1 Bonnie++ Benchmark Sequential I/O Performance

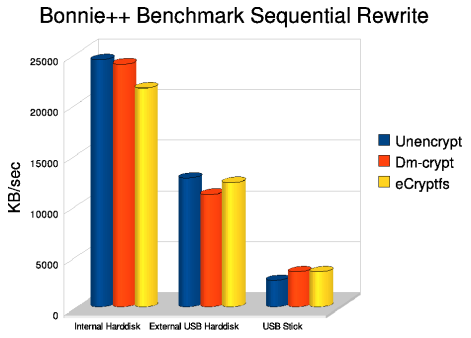
Figure 4.7 shows the Sequential I/O performance for each tool using three different types of hard disk. Obviously, the internal hard disk has the best I/O performance compared to the USB devices. In addtion the external USB hard disk performs better than the 8 GB USB stick. The following figures give the Bonnie++ sequential I/O results.



(a) Sequential Write



(b) Sequential Read



(c) Sequential Rewrite

Figure 4.7: Bonnie++ Benchmark Sequential I/O Performance

Overall, Figures 4.7(a) and 4.7(c) indicate that Dm-crypt encryption has better sequential write and re-write operation performance than eCryptfs encryption, which means that Dm-crypt has less performance penalty compared with eCryptfs encryption.

For the internal hard disk, the Bonnie++ sequential I/O results diverge most for the write operation, where Dm-crypt and eCryptfs have noticeable performance penalties. For the other 2 operations, their performance is quite

close to the unencrypted results. The results for the external USB hard disk are similar. The two encryption schemes give results that differ only a small amount from the unencrypted state.

The write results for the USB stick are essentially the same for all three environments. The two encrypted environments also perform slightly better than the unencrypted state for the rewrite benchmark suggesting that caching is more of a factor in those environments.

The read results for all three disk types are somewhat unexpected in that the encrypted environments perform better than the unencrypted state in almost all cases. The gap between them also increases for the slower I/O media. This result is difficult to rationalize and comes about from the internal functioning of the benchmark suite.

4.4.2 Bonnie++ Benchmark Random Seek Performance

Figure 4.8 reflects the random seek rate of the unencrypted state, Dm-crypt, and eCryptfs based on Bonnie++ benchmark testing. The unencrypted has the highest random seek rate, Dm-crypt ranks second, and eCryptfs has the lowest random seek rate. Once again, this graph illustrates that Dm-crypt encryption has a smaller performance penalty than eCryptfs encryption. The largest penalty comes for the external USB hard disk.

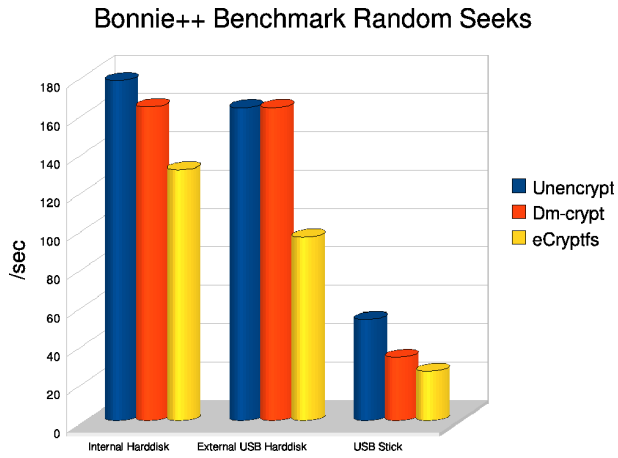


Figure 4.8: Bonnie++ Benchmark Random Seek Performance

4.4.3 Bonnie++ Sequential Block Operation CPU Percentage

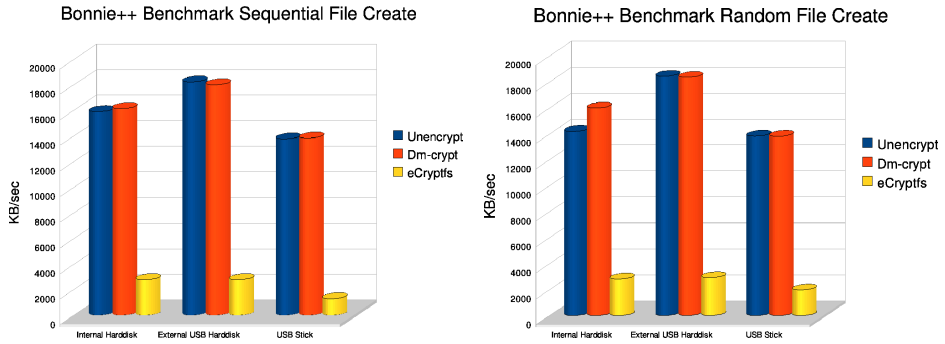
Table 4.1 displays the CPU percentage used for the sequential block operations benchmarks for the Linux encryption tools. Bonnie++ was run on an otherwise idle system. The results from Bonnie++ tests on the internal hard disk show that Dm-crypt encryption does not take much more CPU than the unencrypted situation. In contrast, eCryptfs encryption takes much more CPU resources than both Dm-crypt encryption and the unencrypted state. Similar trends were observed for the other hardware situations.

Tools	Write	Re-write	Read
Unencrypted	13%	0%	0%
$Dm-crypt$	12%	1%	0%
$eCryptfs$	92%	49%	47%

Table 4.1: Bonnie++ Sequential Block Operation CPU Usage

4.4.4 Bonnie++ Benchmark File Create Operations Performance

The Bonnie++ benchmark also tests both sequential and random file create operations for the Linux encryption tools under the three hard disk environments. Figure 4.9 presents these results. It indicates that for both sequential and random operations, the performance of Dm-crypt encryption does not have much difference with the unencrypted state, while eCryptfs encryption has a large performance penalty. These results are consistent with those for most of the other Bonnie++ operations.



(a) Sequential File Create (b) Random File Create

Figure 4.9: Bonnie++ Benchmark File Create Operations Performance

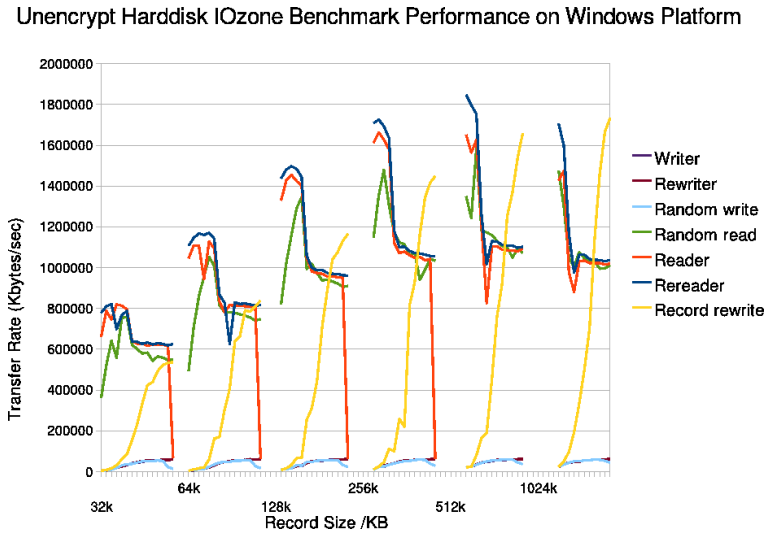
4.5 IOzone Benchmark Results

The IOzone benchmark was run under both Linux and Windows using the internal hard disk. IOzone produces a very large amount of data. In order to analyze its results in an effective way, the most important data was chosen and is presented in separate sections.

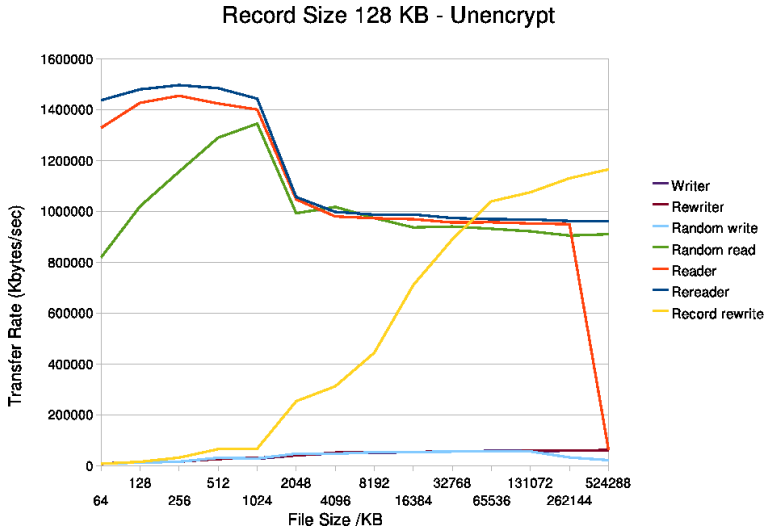
4.5.1 Windows IOzone Benchmark Performance

Figure 4.10 through 4.13 present results for the different Windows encryption facilities for all 7 of the IOzone operation modes. In each case, there is one graph showing the complete results and another graph displaying those for a single record size.

Figure 4.10(a) presents the IOzone benchmark performance for the unencrypted internal hard disk situation, where the record size changes from



(a) Unencrypted

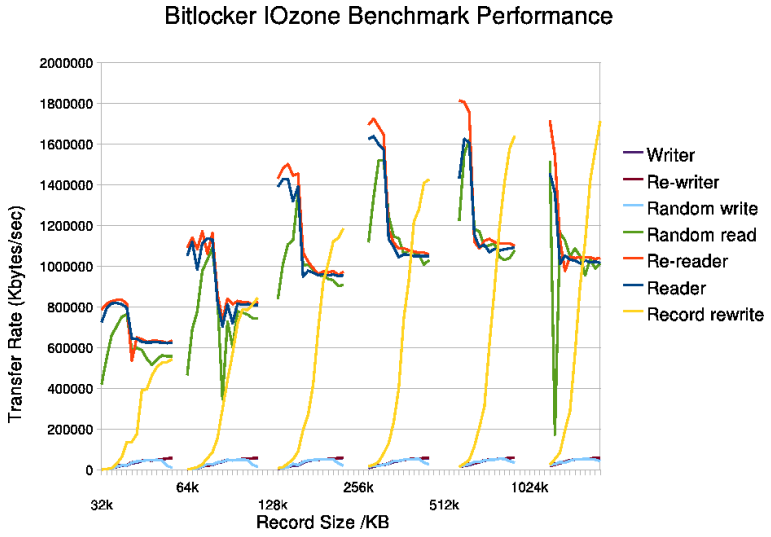


(b) Record Size 128KB

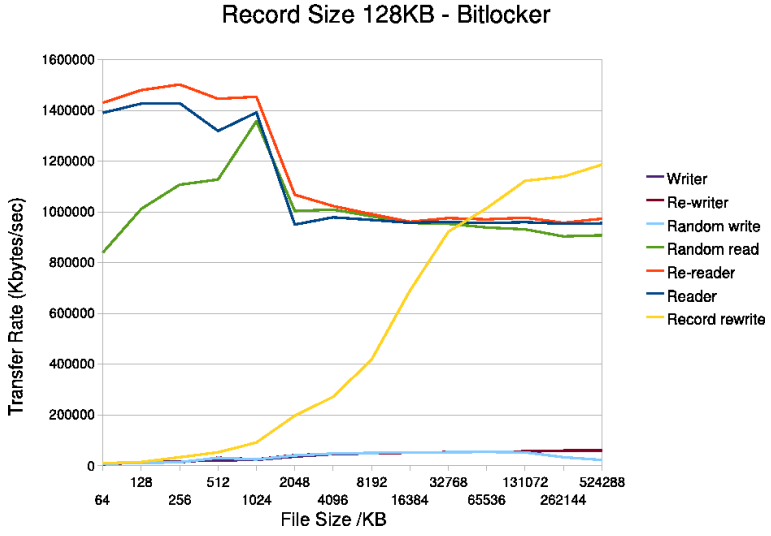
Figure 4.10: Unencrypted Hard Disk IOzone Performance (Windows)

32KB to 1024KB, and, for each record size, the file size increases from 64KB to 1024MB. As the record size increases, the transfer rate slowly increases for each file size. Figure 4.10(b) gives detailed view for the record size of 128KB. The patterns and trends in this example are quite similar for the other record sizes.

These results indicate that the transfer rates of the Rereader, Reader and Random read operations are the fastest, and the transfer rate of the Record rewrite operation increases with the file size, and the other operation modes have a very slow transfer rate.



(a) Bitlocker



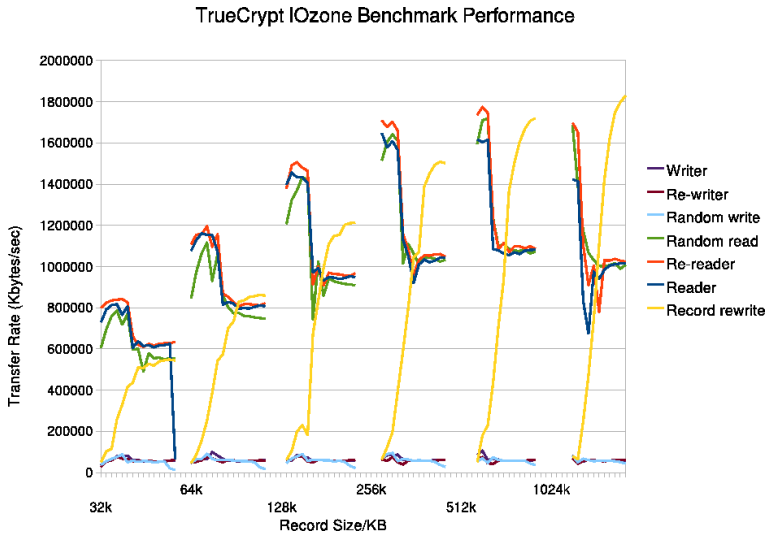
(b) Record Size 128KB

Figure 4.11: Bitlocker Encryption IOzone Performance (Windows)

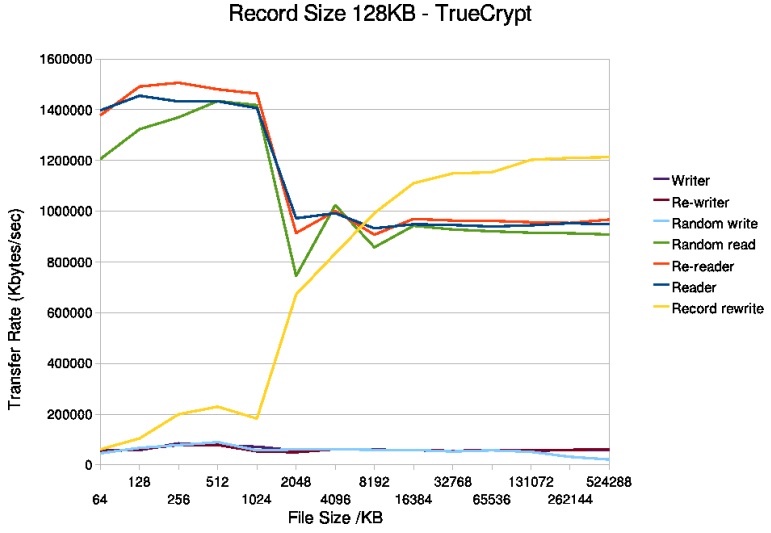
Figure 4.11 displays IOzone benchmark performance using Bitlocker encryption tool using the same record file size ranges. The transfer rates of the

Rereader, Reader and Random read operations are still the fastest, the transfer rate of the Record rewrite operation again grows with the file size. The other operations are very slow. As for the unencrypted state, larger record sizes yield higher transfer rates.

Figure 4.12 shows IOzone benchmark performance using the TrueCrypt encryption tool and the same range of file record sizes. The transfer rates of the Rereader, Reader, Random read and Record rewrite operations are higher than the other 3 operations. For each record size, the transfer rate increases with the file size, and the larger the record size, the higher the transfer rate.



(a) TrueCrypt

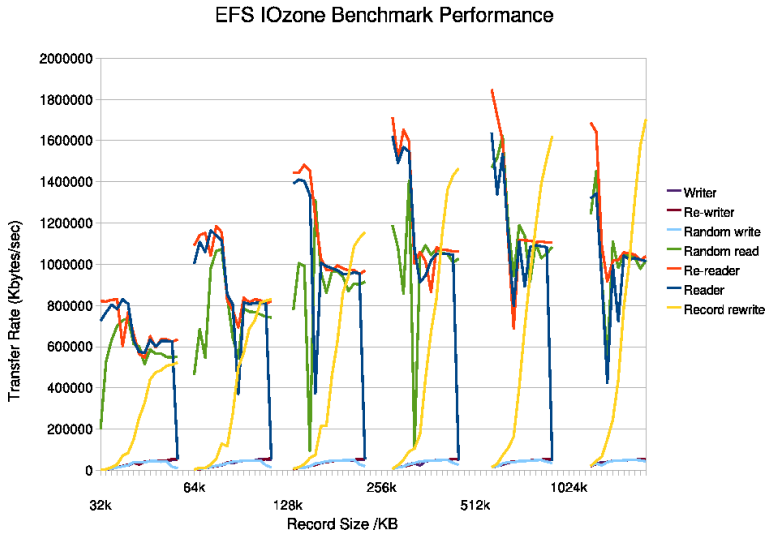


(b) Record Size 128KB

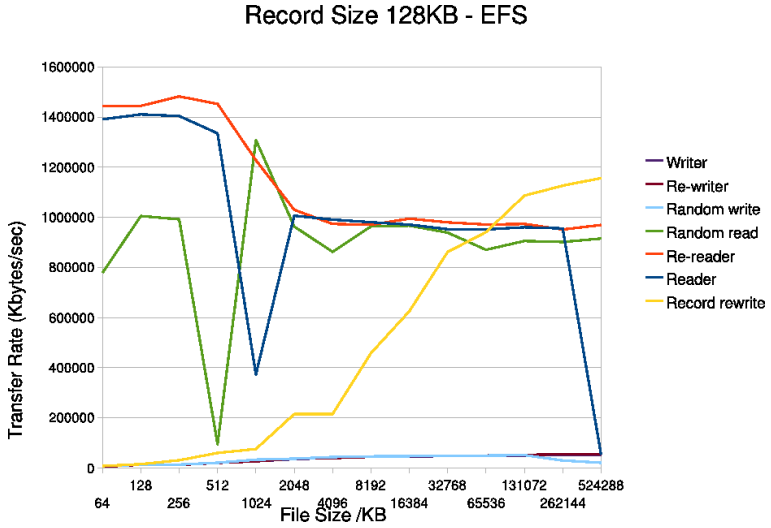
Figure 4.12: TrueCrypt Encryption IOzone Performance (Windows)

Figure 4.13 shows the IOzone benchmark performance for a folder en-

crypted by EFS on the internal hard disk. Again, the transfer rates of the Rereader, Reader, Random read and Record rewriter operations are high, while the transfer rates of other 3 operations are low. For each record size, with the increase of file size, the Record rewrite transfer rate increases consistently, while the Rereader, Reader, Random read operations are not very stable: they oscillate dramatically at the low end of the record size range for all file sizes. The Writer, Rewriter and Random write operations remain slow.



(a) EFS

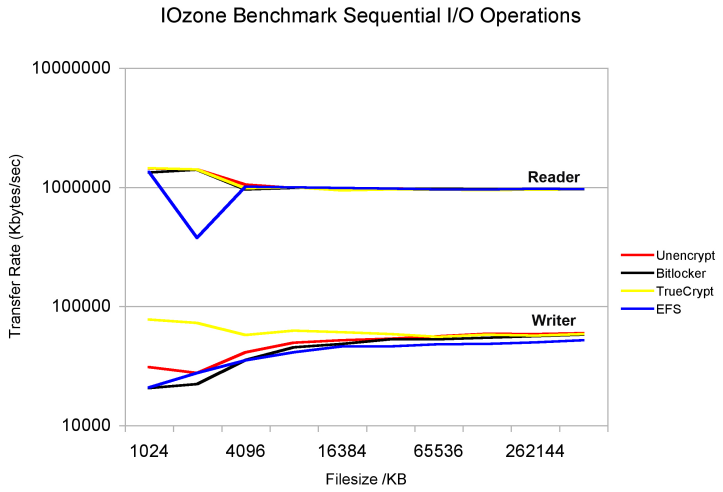


(b) Record Size 128KB

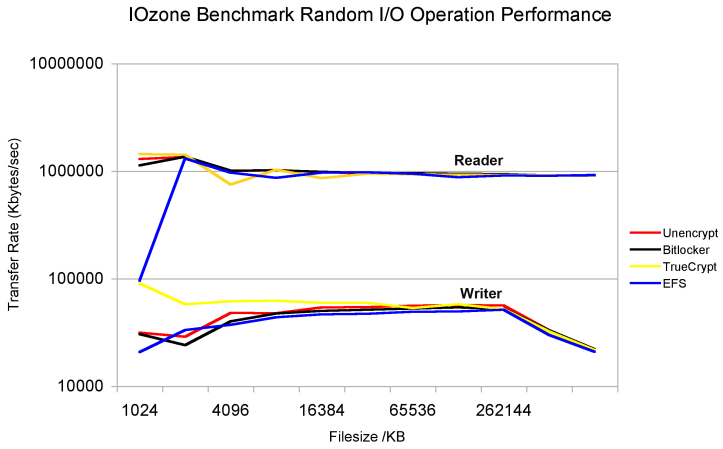
Figure 4.13: EFS Encryption IOzone Performance (Windows)

In generally, for the Windows encryption facilities, the IOzone benchmark performance results shows that the operation modes of Rereader, Reader and Random read (considered as a read operation in general) have higher transfer rates than Re-writer, Writer and Random write (considered as write opera-

tion in general). Record rewrite measures the performance of writing and rewriting a particular spot within a file, and different sizes of the spot will make the performance very different. Here, the results show that the transfer rate of Record rewrite operation for TrueCrypt and EFS all increase with the file size. Figure 4.14 compares the IOzone results for the various Windows cases for the sequential and random read and write operations.



(a) Sequential



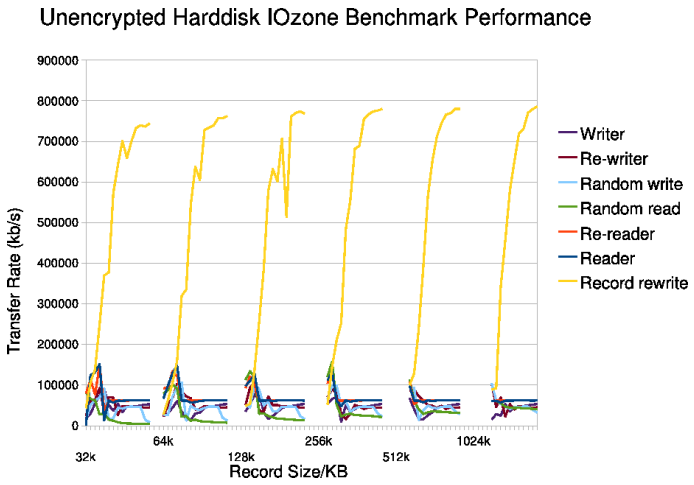
(b) Random

Figure 4.14: IOzone Performance Comparison under Windows

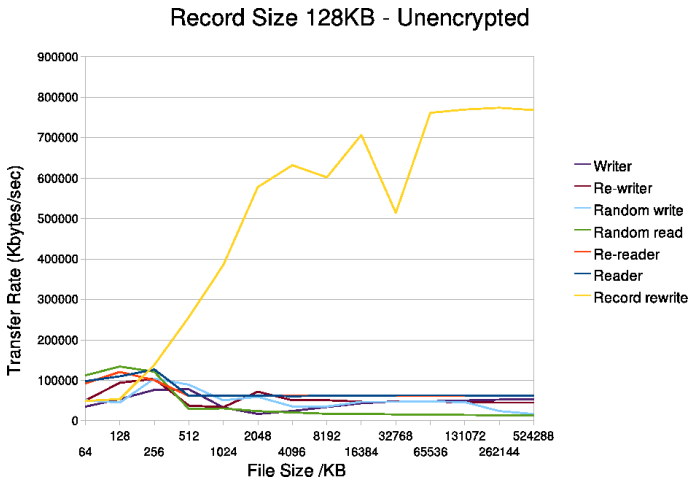
Figure 4.14 shows that the performance on this benchmark varies very little across the different environment. EFS exhibits an odd performance dip for the 1MB file size for both read operations. For the write operations, there is a performance increase starting at the 4MB file size with exception of TrueCrypt, which achieved the higher performance level on smaller file sizes as well.

4.5.2 Linux IOzone Benchmark Performance

IOzone benchmark results for the Linux encryption facilities (unencrypted, Dm-crypt, eCryptfs) IOzone benchmark results are shown in Figures 4.15 through 4.17. All tests were run on the internal hard disk using record size from 32KB to 1024KB and file sizes from 64KB to 1024 MB.



(a) Unencrypted

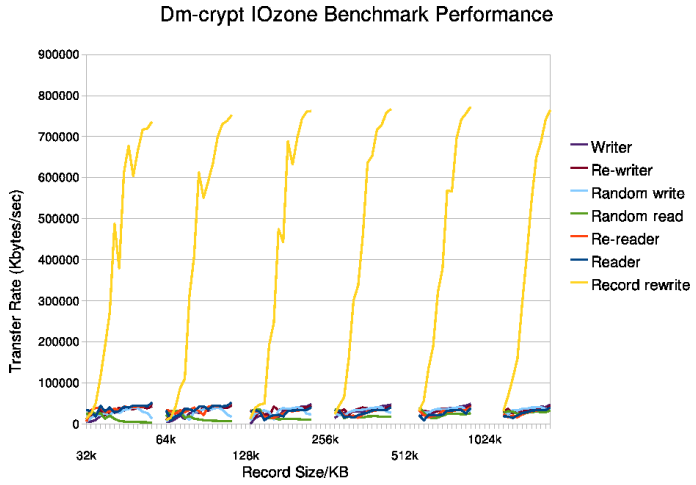


(b) Record Size 128KB

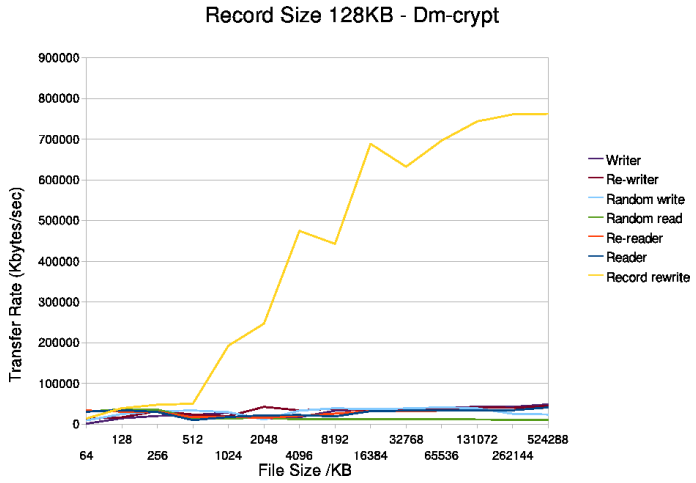
Figure 4.15: Unencrypted Hard Disk IOzone Performance (Linux)

For the unencrypted case and Dm-crypt encryption, the trend of different operation modes is quite similar (see Figure 4.15 and Figure 4.16). For each record size, the transfer rate of the Record Rewriter operation is sharply increased as the file size increased, while for the other operation modes, the transfer rates maintain low level, even when the record size increases.

Figure 4.17 presents the IOzone benchmark performance under eCryptfs encryption. In contrast to the unencrypted and Dm-crypt encryption results,



(a) Dm-crypt

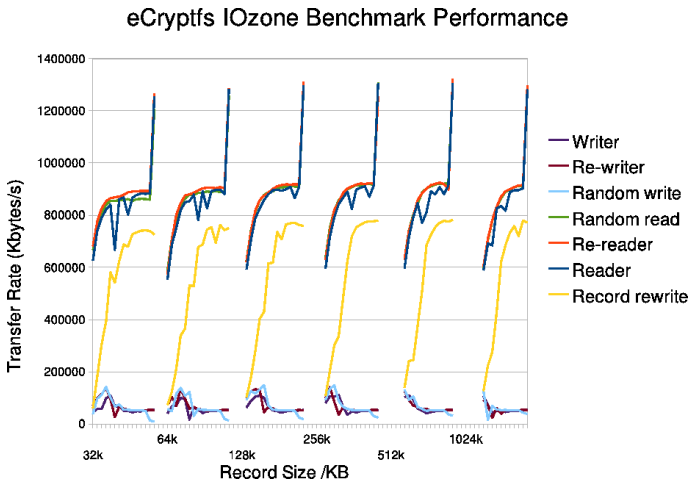


(b) Record Size 128KB

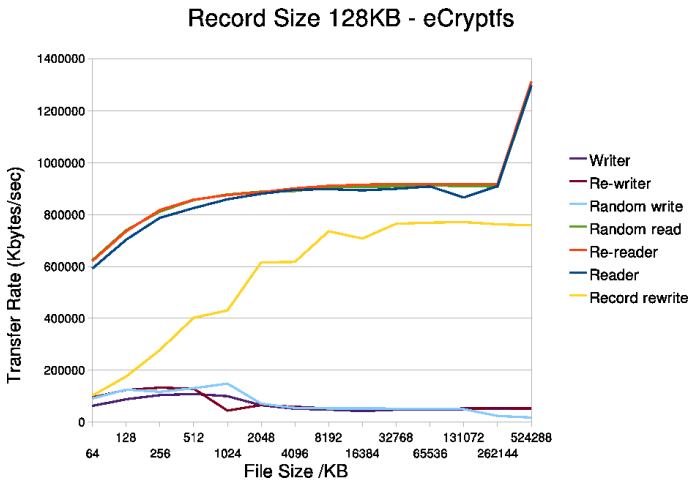
Figure 4.16: Dm-crypt Encryption IOzone Performance (Linux)

the Rereader, Reader, Random read and Record Rewriter operations transfer rates are higher, while the other three operations again show a low speed.

For the Linux environment, IOzone benchmark performance on the internal hard disk, results are quite similar for the unencrypted state and Dm-crypt encryption. The Record rewrite operation mode has largest transfer rate, and it increases with the record size. For all the other operations, transfer rates are very slow and do not change very much. For eCryptfs encryption, the operation modes of Rereader, Reader and Random read (considered as a read operation in general) have higher transfer rates than Re-writer, Writer and Random write (considered as a write operation in general). Also Record rewrite operation has a high transfer rate that increases with the record size.



(a) eCryptfs

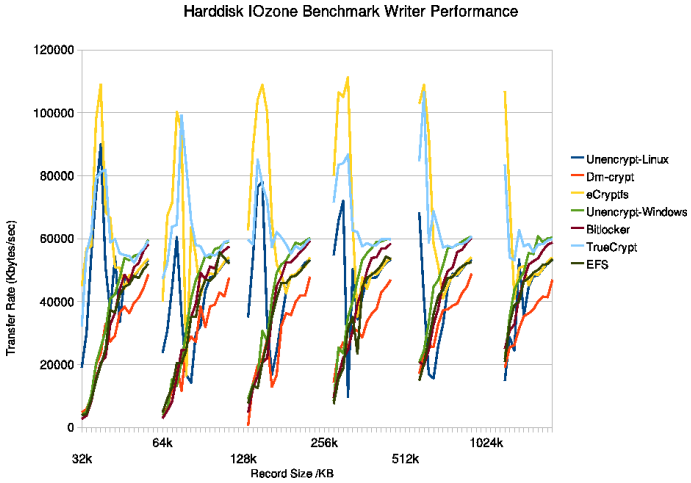


(b) Record Size 128KB

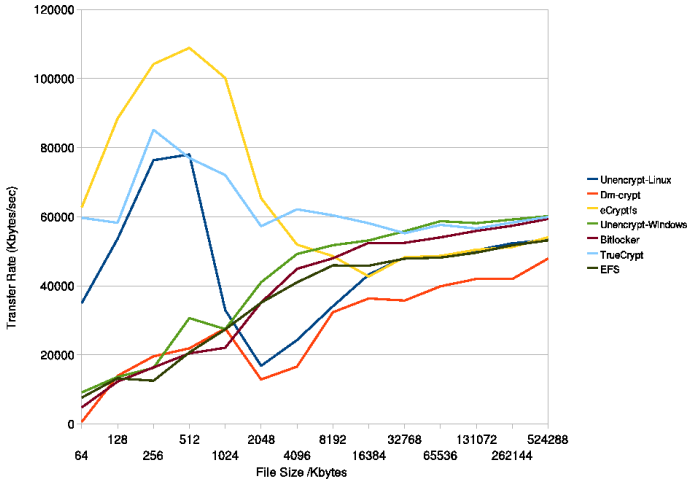
Figure 4.17: eCryptfs Encryption IOzone Performance (Linux)

4.5.3 Three Specific IOzone Benchmark Performance Reports

Figures 4.18 through 4.20 present the results for the operation modes of Writer, Reader and Record Rewrite for all of the tools.



(a) Writer Operation



(b) Record Size 128KB

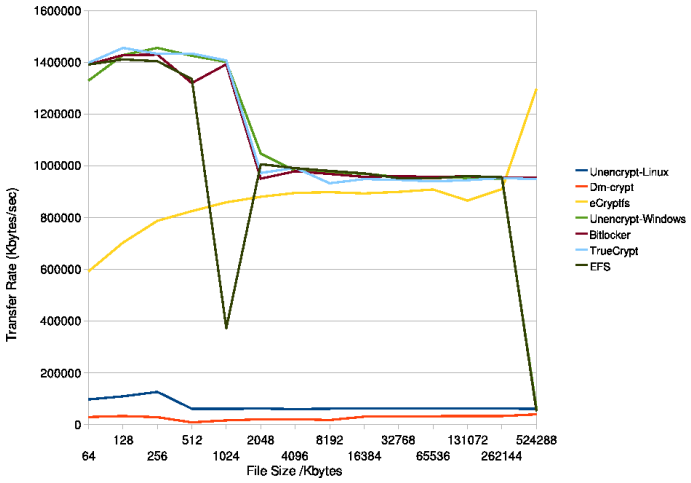
Figure 4.18: IOzone Writer Operation Tools Comparison

Figure 4.18 displays the transfer rates for the IOzone Writer operation for each of the encryption facilities on the same internal hard disk. For the Linux encryption tools (eCryptfs, Dm-crypt, unencrypted), eCrypfs encryption has higher transfer rates than the unencrypt state on Linux, and the transfer rate of Dm-crypt encryption is lower. Still, for the Windows encryption tools (TrueCrypt, Bitlocker, EFS, unencrypted), TrueCrypt encryption has the highest transfer rate, and Bitlocker and EFS encryption are a bit slower than unencrypted state on Windows. For all the tools, the transfer rates of eCryptfs and TrueCrypt encryption rank highest. The other two write operations, Rewriter and Random write, show similar trends.

There is a lot of variation in the Reader results for file sizes smaller than 4-8MB for several tools, including some rather dramatic highs and lows. To some extent, these extremes are likely the result of the specific characteristics for the benchmark and are probably unlikely to be achieved in real user operations as a matter of course.

Considering the results for 8 GB and larger files, the several performance ordering among the tools is (from high to low): unencrypted Windows, TrueCrypt, Bitlocker (all with comparable performance at the highest file sizes), eCryptfs, EFS, unencrypted Linux (these 3 are also very similar to one another), and finally DM-crypt. eCryptfs performans better and Dm-crypt performs worse than they did for the large file sequential write I/O test considered earlier.

(a) Reader Operation



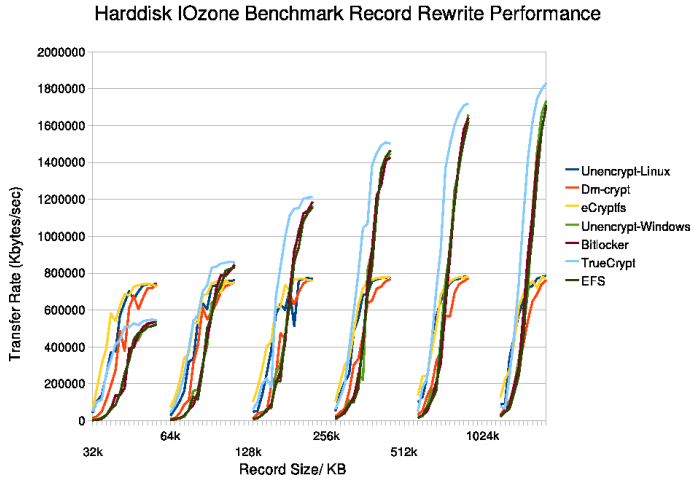
(b) Record Size 128KB

Figure 4.19: IOzone Reader Operation Tools Comparison

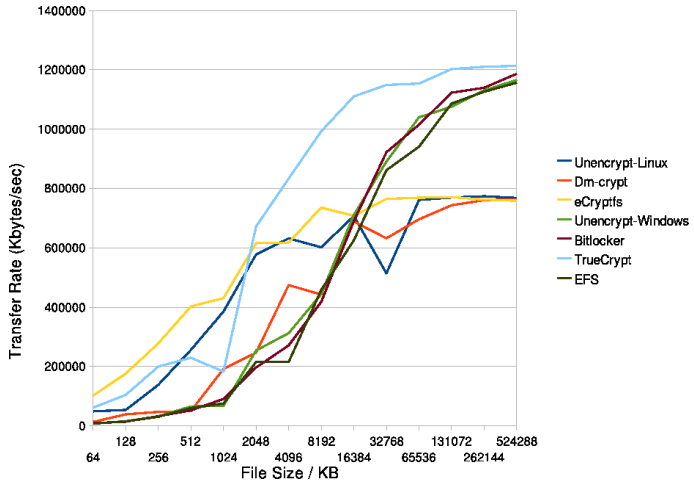
Figure 4.19 presents the the transfer rate of the IOzone Reader operation

for each of the encryption facilities. Generally, the transfer rates of Windows encryption tools is faster than the transfer rate of Linux encryption tools. For Linux tools, the transfer rate of eCryptfs encryption is much faster than Dm-crypt encryption and the unencrypted situation. In contrast, the transfer rates of Windows encryption facilities are quite close to one another. The other two read operations Rereader and Random read are similar to this trend.

These Linux results are quite anomalous in two ways: the extremely high performance of eCryptfs, and the extremely low performance of Dm-crypt and the unencrypted Linux environment. These results seem suspect and all into question the accuracy of IOzone in their Linux environment in general. The IOzone results are repeatable across multiple runs and similar trends are observed in other devices.



(a) Record Re-write



(b) Record Size 128KB

Figure 4.20: IOzone Record Rewrite Operation Tools Comparison

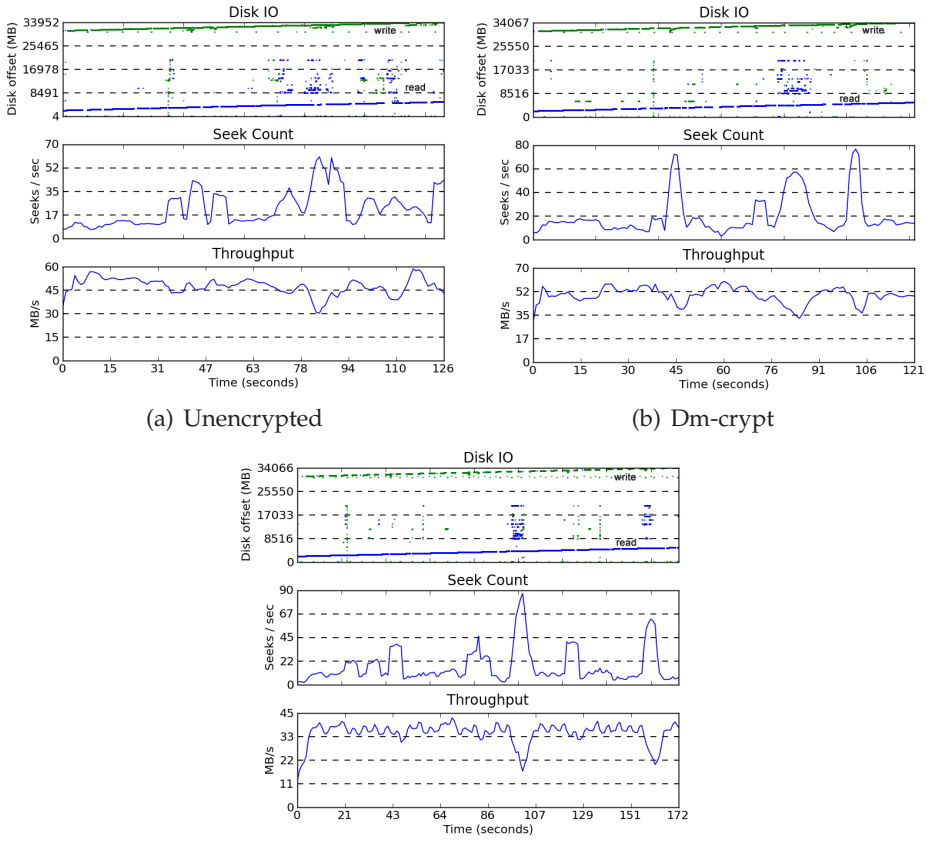
Figure 4.20 shows the IOzone benchmark performance Record rewrite results which measure the performance of writing and re-writing a particular

spot within a file. The graph shows that for all the encryption approaches, the transfer rate grows quickly as the record size increases. In addition, for a given record size, the transfer rate increases with the file size. For the larger the record size, the transfer rates of TrueCrypt, EFS and Bitlocker encryption increase faster than other tools.

4.6 Seekwatcher I/O Results

Seekwatcher graphs the raw I/O data collected by blktrace during a specified I/O operation. For these tests, the same commands were run as for the Linux sequential I/O tests presented in section 4.3.

Figure 4.21 is the results of the Seekwatcher benchmark for the 3G file write operation on the same internal hard disk partition. The three Disk I/O graphs show that the write operation is based at the same disk offset and thus uses the same portion of the partition.



(a) Unencrypted

(b) Dm-crypt

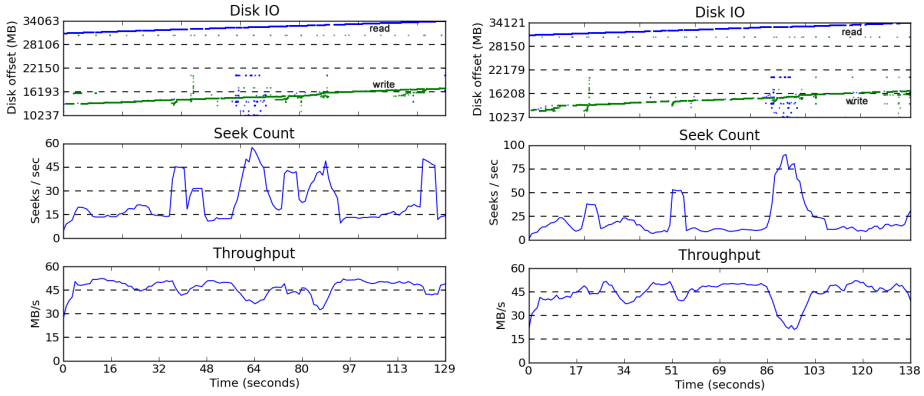
(c) eCryptfs

Figure 4.21: Seekwatcher I/O Write File Operation (Linux)

However, the curves for Seek Count and Throughput for Dm-crypt and eCryptfs are very different from the unencrypted situation. This indicates that

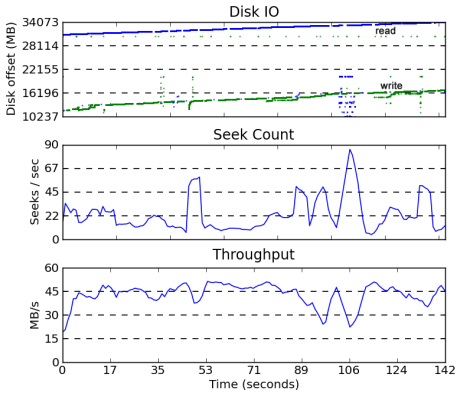
disk head movement pattern is quite different for the unencrypted case. The I/O throughput rates are various in different ways among the 3 environments, with the rate for eCryptfs being significantly lower than the other two.

Figure 4.22 shows the results of the Seekwatcher benchmark for 3G file read operation. As expected, the read operation is based on the same disk offset. Once again, the Seek Count and Throughput partten for Dm-crypt and eCryptfs are very different from the unencrypted situation for the same I/O operation. This time, eCryptfs achieves the same overall performance levels as the other two cases.



(a) Unencrypted

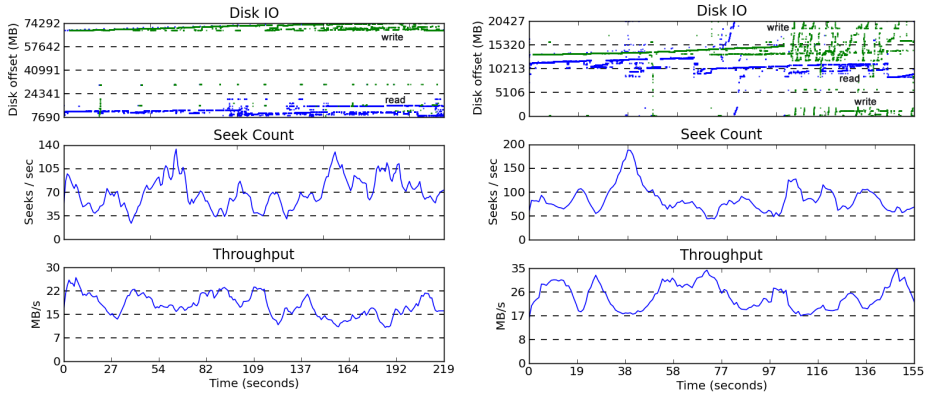
(b) Dm-crypt



(c) eCryptfs

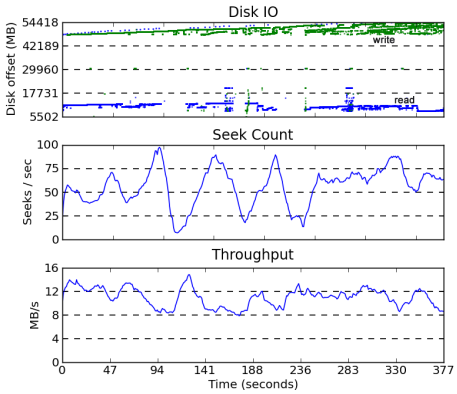
Figure 4.22: Seekwatcher I/O Read File Operation (Linux)

Figure 4.23 shows the results of the Seekwatcher benchmark for the 2.1G directory write operation. The three Disk I/O graphs show that the disk portion used is different in all 3 cases. However, during the course of the same I/O operation, the Seek Count and Throughput profiles for Dm-crypt and eCryptfs are very different from the unencrypted situation. The throughput for eCryptfs lies in the range of 40% to 60% of that achieved for the other two cases.



(a) Unencrypted

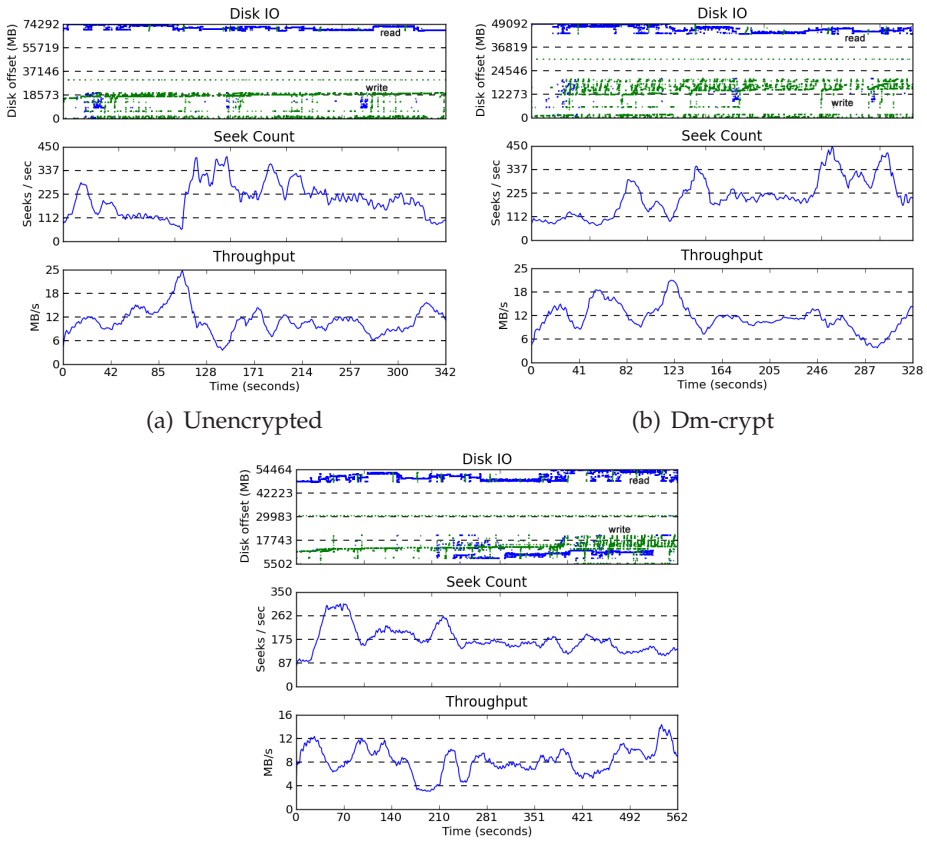
(b) Dm-crypt



(c) eCryptfs

Figure 4.23: Seekwatcher I/O Write Directory Operation (Linux)

Figure 4.24 shows the results for the Seekwatcher benchmark for the 2.1G directory read operation. The directory read operation disk portion used is different in the 3 cases. Once again, the Seek Count and Throughput profiles for Dm-crypt and eCryptfs are differ from the unencrypted situation. Interestingly, eCryptfs has a much smoother as less varying seek count profile than the other 2 cases. It also exhibits lower overall throughput over the period of the operation.



(a) Unencrypted (b) Dm-crypt

(c) eCryptfs

Figure 4.24: Seekwatcher I/O Read Directory Operation (Linux)

The Seekwatcher throughput results are consistent with the averaged results seen in section 4.3. Seekwatcher provides a detailed view of how the throughput and disk access rate varies over the course of each operation. There are major differences in these profiles for the three environments.

4.7 HPC Sample Calculation Results

In HPC calculation experiment, Gaussian 09 executed the same simulation under the various encryption environments on both Windows and Linux platforms. Figure 4.25 presents the real and I/O times for each case; and also

calculates the performance penalty of different encryption approaches for the large random I/O operations performed during the calculation.

Tests	Sequential Access					Random Access				
	I/O Operation Size			Multi File	Directory Hierarchy	Seek Rate	I/O Operation Size			Multi File
	Small <10MB	Medium	Large >100MB				Small	Medium	Large	
Sequential I/O		Y	Y	Y	Y					
Bonnie++	Y	Y		Y		Y	Y	Y		Y
IOzone	Y					Y	Y			
Seekwatcher			Y	Y	Y					
HPC Calculation									Y	

Figure 4.25: Gaussian 09 - HPC Sample Calculation Comparison

These results show that Linux encryption tools (Dm-crypt, eCryptfs) have a larger performance penalty than the Windows encryption tools (Bitlocker, TrueCrypt and EFS). Moreover, eCryptfs encryption has the biggest performance penalty and takes much more I/O time than others. For Window encryption approaches, the performance penalty of EFS is bigger than TrueCrypt and Bitlocker encryption.

The Windows environment itself imposes a slight performance penalty. The unencrypted state Windows calculation took about 20% longer than the same Linux calculation on its unencrypted disk.

Chapter 5

Evaluation and Discussion

This chapter will cover the analysis and discussion of the experimental results. Since the goal of the thesis is to evaluate the benefits of different encryption facilities to normal users, it will discuss each tool from different dimensions that users care about based on these results.

5.1 Costs and Platform Compatibility

Figure 2.3 presented an overview of different encryption facilities, including the licensing and running platforms of each tool.

Bitlocker and EFS are Microsoft proprietary software and they can only be used on Windows versions. EFS can be enabled in most Windows versions, such as Windows 2000/XP/Vista/7, and Windows Server 2003/2008. However, Bitlocker is only available on Windows 7/Vista Ultimate and Enterprise and Windows Server 2008. Currently, major hardware vendors charge extra money for Bitlocker (the default is no Bitlocker), but this may change in the future as more and more users will require this feature.

TrueCrypt is a free open source software for multiple platforms. It supports Linux platforms with kernel 2.6.5 or higher, Windows 7/Vista/XP and Mac OS X.

eCryptfs and Dm-crypt both use the GNU general public license, and they can be used on Linux platforms. eCryptfs can be running with Linux kernel 2.6.19 and higher; Dm-crypt is available on Linux kernel 2.6 and higher.

Obviously, if normal users consider only the cost and platform compatibility factors, TrueCrypt is preferred. In addition, Windows users can also think about Bitlocker and EFS, and Linux users can try to use eCryptfs and Dm-crypt.

5.2 Ease of Installation and Use

Details about setting up each encryption approach were discussed in section 3.3.

Currently, only Bitlocker, EFS, TrueCrypt have GUI interface and navigation for setting up encryption. eCryptfs and Dm-crypt encryptions can just be implemented via the command line. Dm-crypt uses the interface cryptsetup to initialize the encryption; also, it can work with LUKS support. eCryptfs can be considered as an cryptographic filesystem.

In general, as long as the function and principle of each encryption approach is understood, all these encryption tools are easy to install and set up. None of them are that difficult to set up by following the navigation or the user guide.

Another thing that normal users may care about is the time to set up the encryption. Figure 4.1 presents the approximate time of each tools to initialize encryption on the same 8G USB stick. The result shows that EFS takes about 48 minutes, Bitlocker takes 37 minutes, TrueCrypt needs 18 minutes, and eCryptfs and Dm-crypt take less than 3 minutes. Although that is just an approximate timing based on a 8 GB USB stick, it still can give users a primary impression about the time cost of each facilities. These timing values represent the worst case scenario as I/O is slowest to this device.

5.3 Features and Effectiveness Analysis

All of the encryption tools discussed in the thesis have the functions required provide to make automated, user-transparent encryption on different types of storage media. Bitlocker, Dm-crypt, and TrueCrypt are block device encryption approaches (Bitlocker is implemented as full disk encryption), while, EFS and eCryptfs are filesystem-level encryption tools.

Using block device encryption, Dm-crypt, TrueCrypt and Bitlocker can provide additional security beyond existing OS security mechanisms, and can protect the data even the physical device been removed from the system. The attackers can not determine the filesystem and directory structure of the encrypted devices unless they have the encryption key.

Moreover, Bitlocker can use encryption on the booting disk, and Windows 7 has the new feature of Bitlocker to go which can also encrypt USB devices. Truecrypt can create encrypted file containers, encrypted non-system volume partitions, and versions later than 5.0 also include system partition encryption on Windows systems. However, block device encryption approaches are not convenient when resizing the encrypted partition after initialization.

EFS and eCryptfs, as approaches to filesystem-level encryption, have flexible file-based key management, and each file can potentially be encrypted with a separate encryption key. Thus, encrypted files can be managed independently. Also, these approaches can use public key cryptography to enforce the access control. But one limitation of them is that they can not encrypt the file system metadata, such as the directory structure, file names, sizes or timestamps. This can be problematic if the metadata itself needs to be kept secret.

5.4 Performance Penalty Evaluation

The following evaluations begin with baseline analysis in the absence of encryption and then go on to compare the various encryption options.

5.4.1 Storage Media Comparisons Without Encryption

An internal hard disk, external USB hard disk and USB stick are all considered in this thesis' experiments. Normally, the performance of hard disk will depend very much on the types and features of the hard disk device.

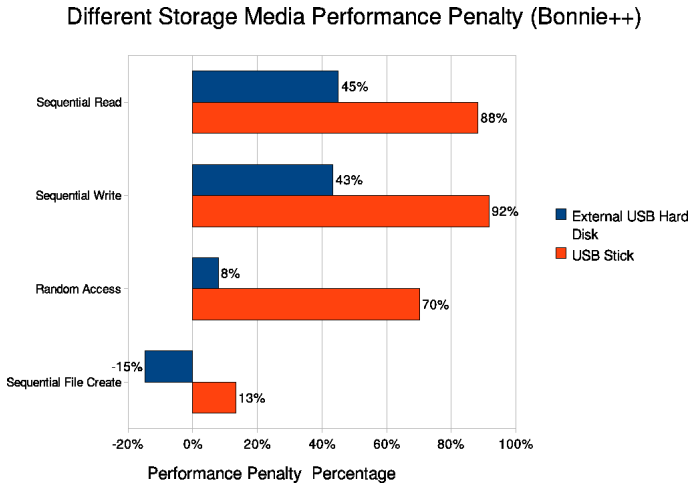


Figure 5.1: Storage Media Performance Comparison (Unencrypted Linux)

Figure 5.1 shows the performance penalty of the external USB hard disk and USB stick with respect to the internal hard disk. For most cases, both external devices used in the experiment have a significant performance penalty compared to internal hard disk, especially, the USB stick has a larger penalty.

Figure 5.2 presents the USB stick performance penalty compared to the internal hard disk on the Windows platform for different I/O operations. The USB stick has a large performance penalty compared with the internal hard disk. The sequential directory write operation reaches a 95% penalty percentage, the worst observed case, indicating that it takes almost twice as long as for the internal hard disk. These results are generally comparable to the Linux results.

5.4.2 Operating System Comparison Without Encryption

Another factor supposed to be considered is the operating system independent of encryption. Figure 5.3 shows the Linux performance penalty with respect to Windows for a variety of I/O operations using the internal hard disk. In Figure 5.3, for the different I/O operations, Linux has performance penalty percentage ranging from 16% to 154% compared to Windows. In other words,

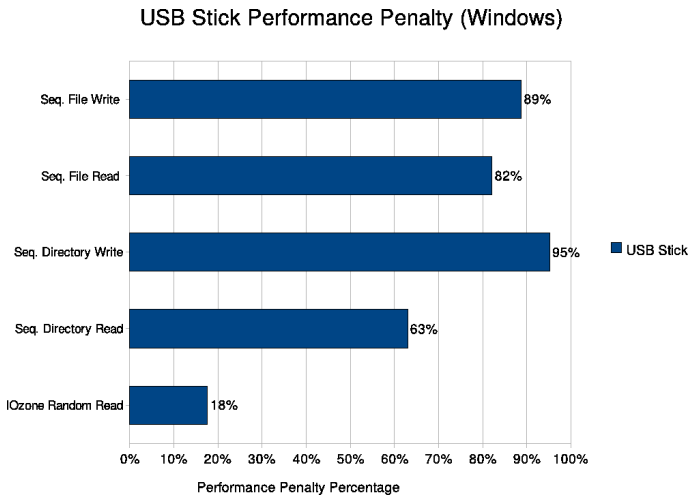


Figure 5.2: USB Stick Performance Penalty on Windows OS

for the same I/O operations on the same hard disk, the Windows 7 Ultimate environment performs better than Red Hat 5.4 with the Linux kernel 2.6.18 environment. The sequential directory write operation and IOzone's small to medium random write operation show the least performance difference.

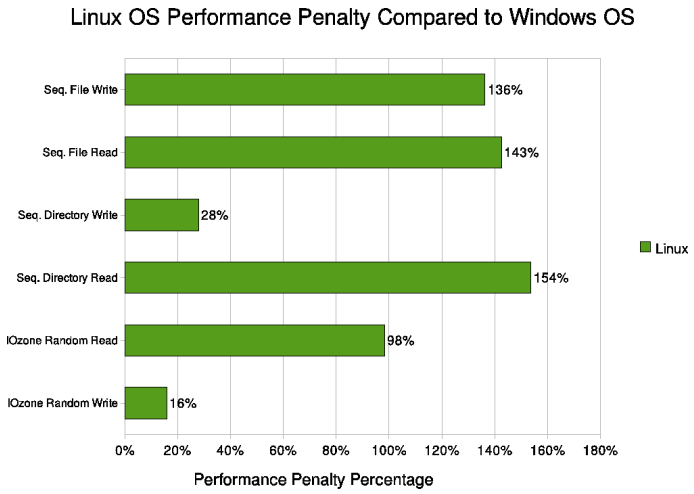


Figure 5.3: Operating System Performance Comparison (Unencrypted)

5.4.3 Encryption Strategies

The two main encryption strategies discussed in the thesis are used by most of the popular approaches nowadays. Block device encryption apples to the block device layer such as a hard disk partition or even a whole volume. Bitlocker, TrueCrypt and Dm-crypt are all block device encryption tools. Filesystem level encryption encrypts data at the filesystem level. eCryptfs can be considered as a kernel-native cryptographic filesystem in the Linux system.

Also EFS is implemented by the filesystem-level EFS function on the Windows system. So these two approaches implement on the filesystem level encryption strategy.

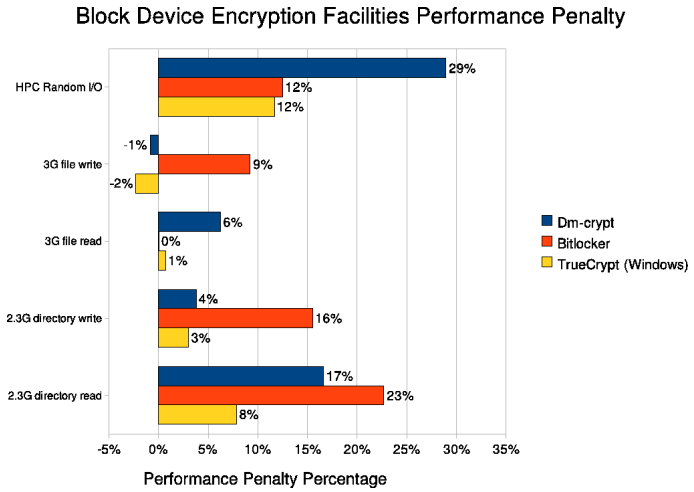


Figure 5.4: Block Device Encryption Approaches Performance Penalty

Figure 5.4 shows the performance penalty of different block device encryption approaches with respect to performance on an unencrypted device using the internal hard disk on the same platform. For the HPC random I/O, Dm-crypt has a larger penalty than Bitlocker and TrueCrypt. However, for the large sequential I/O operations, Bitlocker has a larger penalty than Dm-crypt and TrueCrypt. The largest penalty is 29% while most of the values are less than 20%.

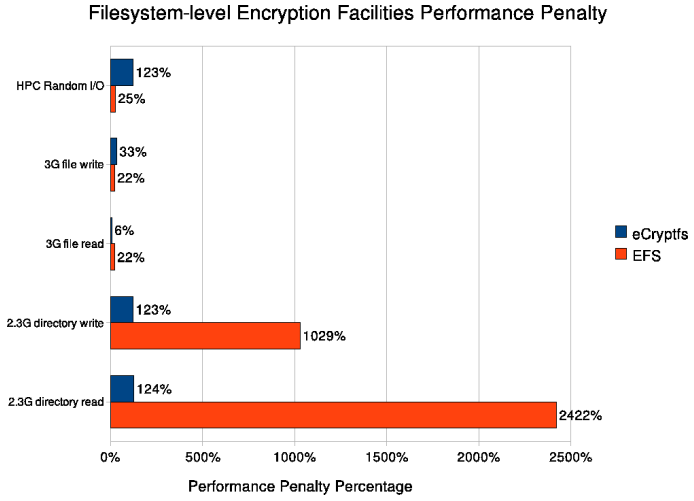


Figure 5.5: Filesystem-level Encryption Approaches Performance Penalty

Figure 5.5 reflects the filesystem-level encryption tools' performance penalty with respect to the unencrypted state on the internal hard disk. EFS has a much larger penalty than eCryptfs, especially for the directory I/O operations where

it is catastrophically poor. However, eCryptfs also often imposes a significant penalty of more 2 times.

5.4.4 Multi-platform Approach: TrueCrypt

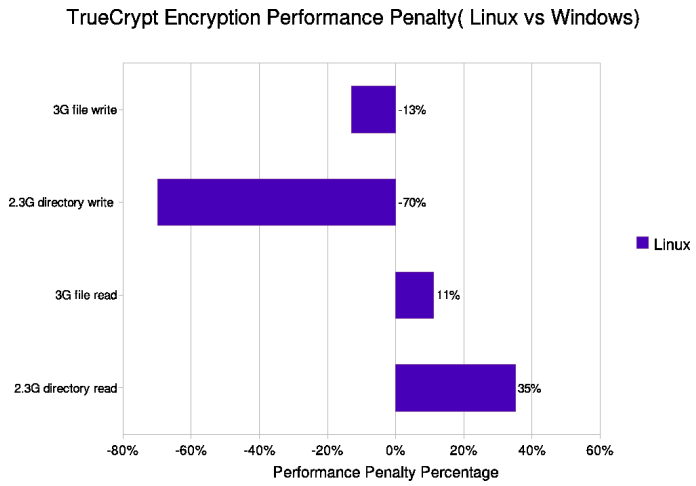


Figure 5.6: TrueCrypt Performance Penalty (Windows vs Linux)

TrueCrypt is a free open-source encryption tool on multiple platforms. Figure 5.6 presents the sequential I/O performance penalty for TrueCrypt on both Windows and Linux systems. The results shows that for sequential write operations, TrueCrypt on the Linux system has better performance than Windows, however, the results of sequential read operations are opposite.

5.4.5 CPU Usage

Both the results from Bonnie++ and HPC sample calculations show that for Linux encryption facilities, eCryptfs takes much more CPU time when executing I/O operations, which is caused by kernel service operation; while Dm-crypt performs as well as unencrypted situation and takes very little extra CPU time.

5.5 Recommendations

Based on the results in this thesis, the following suggestions are given to normal users, especially new users to the encryption facilities.

5.5.1 Scenario 1: Laptops

For laptop users, it is very necessary to use encryption for data security. The number of laptop users are increasing, but laptops are easily lost or stolen.

Block device encryption tools are recommended for laptop users as they encrypt all data on the hard disk.

For Window users, Bitlocker is also a good choice, especially nowadays, as most laptops have TPM support, which can help Bitlocker work better. Although Bitlocker is only available on Windows Vista/7 Ultimate and Enterprise, it is worth spending extra money for this feature. Also, it is not very difficult to update an existing Windows Vista/7 Home Basic operating system variant to Ultimate version in any significant way. It can be easily selected during OS installation, and it is also easy to enable later.

For Linux users, Dm-crypt is suggested. It is easy to insall with kernel 2.6 or higher, and also it does not affect hard disk I/O operation performance.

TrueCrypt is a good choice for external media as it is free open source software. In addition, external devices using TrueCrypt can be used on multiple platforms. Once Truecrypt encrypts a hard disk partition under one system, the encrypted partition can be automatically mounted on another system.

5.5.2 Scenario 2: Desktops

Traditional computers with large amounts of sensitive data are given the same suggestions as laptops.

5.5.3 Scenario 3: Limited Sensitive Data

For existing desktop system users, with limited amounts of sensitive data, the filesystem level encryption tools can meet the requirements.

For Windows users, EFS can be a good method. The EFS attribute can be enabled for files or folders which are important. Alternatively, EFS can encrypt an empty directory, where confidential data can be stored as need. However, if there is a large amount of data to be encrypted, EFS will be very time-consuming.

Linux users can choose eCryptfs for encryption of existing file systems, which takes only a very short time to set up. However, large directory I/O operations will takes more time than previously. If you need to copy or move the encrypted directories quite often, then eCryptfs is not a good choice.

.

5.5.4 Scenario 4: External Media

If you want to encrypt an external USB stick or hard disk, TrueCrypt certainly works fine for both Windows and Linux users. For users with Windows 7 Ultimate or Enterprise OS, Bitlocker to go also functions very well. However, EFS can not encrypt the folders or files on external USB devices.

For Linux users, both eCryptfs and Dm-crypt can be used for encryption on external USB devices. In case the USB device is lost or stolen, Dm-crypt is more secure.

5.6 Problems Encountered

1. When setting up TrueCrypt encryption, Red Hat 5.4 with kernel 2.6.18 can not initialize any Truecrypt-6.3a file systems due to an acknowledged kernel bug. One solution it to upgrade the current kernel to 2.6.24 or later, but this is not practical for ordinary users. So in the experiment, instead of using Red Hat platform for TrueCrypt encryption, Ubuntu 8.10 platform with kernel 2.6.24-27-generic was used.

2. In the sequential I/O performance tests, the memory cache makes a big difference in the real time of I/O operations. To make sure the results are not affected by cache, flushing cache was an important step in each test.

3. The benchmark tool HD Tune Pro was tried on the Windows platform. However, it is limited to measuring the lowest level disk operations, and thus could not show the performance effects of encryption.

4. The IOzone benchmark results of the Linux encryption facilities are quite anomalous. eCryptfs shows extremely high performance, but Dm-crypt and the unencrypted Linux environment have extremely low performance. These results seem suspect and call into question the accuracy of IOzone in the Linux environment in general.

5.7 Experiment Validation and Reliability

When trying to compare different approaches, a reference model is needed. In the experiment, the unencrypted state is the reference model for the other encryption approaches. In addition, to compare various factors related to each approach, one factor is made variable while other factors remain stable. Then the result will show the affect of that certain factor on the performance of different approaches.

In the thesis, the aim is to compare different encryption tools. First, they are divided into two groups (Windows and Linux). Each platform has the same hardware environment. Then, to evaluate different benefits of these tools, the same sequential I/O tasks are tested, and also different benchmark tools are used to measure performances for a variety of I/O types. The performance penalty is compared for the different cases.

To eliminate the experiment errors, for sequential I/O tasks, tests are repeated three times each, and results presented are the average. For different benchmark tests, the benchmark tools' internal mechanisms for repeated testing were used to ensure that the results are reliable.

5.8 Future Work

The thesis evaluated different encryption tools with various performance tests.

It would be interesting to perform case studies about these encryption approaches with actual users operations. For example, observing and recording all the operations a user did in a given time period, then executing the same operations under different encryption situations using each tool would be an interesting comparison. Also database operations can be performed to compare the access time under different encryption states using each encryption tool for a very different kind of I/O operation.

Due to the kernel bug on Red Hat 5.4 with kernel 2.6.18 for TrueCrypt 6.3a file systems, TrueCrypt encryption was just compared based on an external USB stick in the thesis. It would be beneficial to continue with TrueCrypt encryption performance comparison on multiple hardware environment and platforms.

As mentioned, IOzone benchmark results under Linux environment are odd. The IOzone benchmark tests were repeated under different hard disk environments, and produced similar trends.. It would be important to study the accuracy of IOzone benchmark in the Linux environment in more detail.

Future work can also consider different types of hard disks. For example, the encryption performance based on other types of hard disk like SCSI might be different.

Furthermore, people can deploy each encryption tools on different platforms, such as cell phones, which are popular, mobile and easily lost or stolen. There will be a strong requirement to have encryption function on such portable devices in the future, and so a similar study would be useful.

Finally, it might be interesting to observe ordinary users attempting to use each of the encryption tools in various environments in order to confirm the thesis' judgment about the tools' ease of use.

5.9 Conclusion

With the aim of comparing different encryption facilities on multiple platforms and evaluating their benefits to the normal users, this thesis has performed research about encryption technologies, described the features and advantages/disadvantages of some popular encryption facilities, and also measured and compared the performance penalty of these facilities.

By demonstrating the installation and use of different encryption tools on different platforms, it provides general information of how to use these tools for encryption. Based on all of these results and evaluations, recommendations for encryption tools were made for several common scenarios. Hopefully, the results of this thesis will be valuable to users who want to use encryption technology.

Appendix A

Experiment Setup

A.1 Dm-crypt Encryption Setup on Logical Volume Without LUKS

```
# cryptsetup create DMCRYPT /dev/mapper/LVMG-DATA
Enter passphrase:

# cryptsetup status DMCRYPT
/dev/mapper/DMCRYPT is active:
cipher: aes-cbc-plain
keysize: 256 bits
device: /dev/mapper/LVMG-DATA
offset: 0 sectors
size: 125575168 sectors
mode: read/write
# ls -l /dev/mapper/
total 0
crw——- 1 root root 10, 63 Feb 10 19:40 control
brw-rw—- 1 root disk 253, 1 Feb 11 08:09 DMCRYPT
brw-rw—- 1 root disk 253, 0 Feb 10 21:27 LVMG-DATA

# mkfs.ext3 /dev/mapper/DMCRYPT
# mount /dev/mapper/DMCRYPT /mnt/crypt/
# df -H
Filesystem Size Used Avail Use% Mounted on
/dev/sdb1 11G 4.1G 5.6G 43% /
tmpfs 1.1G 0 1.1G 0% /dev/shm
/dev/sdc1 512M 7.8M 504M 2% /media/BIANCA
/dev/mapper/DMCRYPT 64G 189M 60G 1% /mnt/crypt

# umount /mnt/crypt/
# cryptsetup remove DMCRYPT
```

A.2 Seekwatcher Installation

- First, download and install blktrace. Make sure blktrace can have live run.

```
# yum install blktrace
# mount -t debugfs debugfs /sys/kernel/debug
# blktrace -d /dev/sdb3 -o - | blkparse -i -
```

- Second, install matplotlib for generating graph by Seekwatcher. On matplotlib website [49], there are detailed infomation about matplotlib instruction. Using EPD32-6.1 [50] is a recommended method here. After installing EPD32-6.1 successfully, do not forget to export the path.

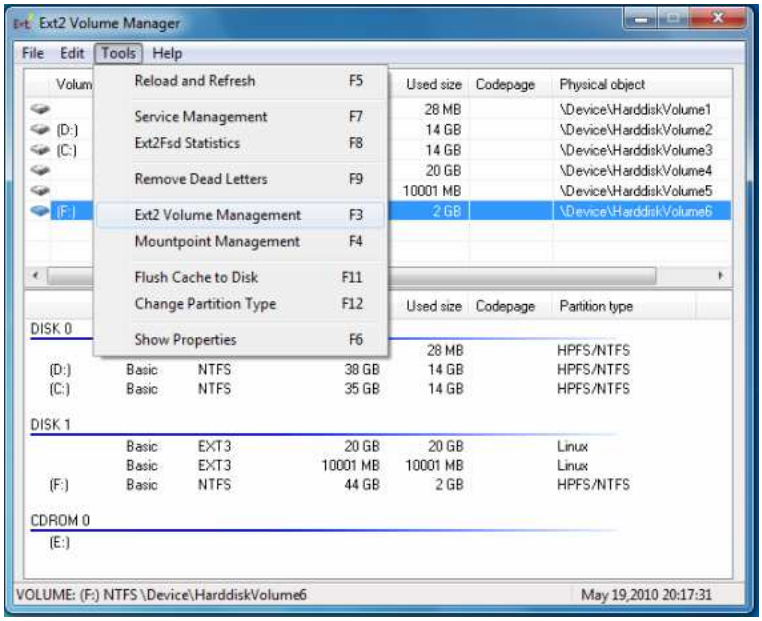
```
# export PATH = "/usr/EPD/bin:" \$PATH
```

- Third, install Seekwatcher version 0.12. To read the output file of Seekwatcher, using the following command:

```
# blkparse -i read_file_unencrypt.trace.blktrace.0
```

A.3 Ext2Fsd – Mount Linux Hard Disk Partition on Windows

Ext2Fsd is an open source Linux ext2/ext3 file system driver for Windows systems. It will allow users to have read/write access to Linux partitions from Windows platform. First choose Linux partition which you want to mount, then change the filesystem to 'NTFS' or 'FAT', after that this partition will be mount as a Windows volume. Here is the graph of volume manager interface:



A.4 HPC Calculation Gaussian 09 Input File

```
%rwf=hpctest.rwf
%chk=hpctest.chk
#p CCSD/6-31G(d,p) Geom=Connectivity Transformation=Full

HPC I/O performance benchmark

0 1
 C
 H          1          B1
 H          1          B2     2          A1
 H          1          B3     3          A2     2          D1     0
 C          1          B4     3          A3     4          D2     0
 H          5          B5     1          A4     3          D3     0
 H          5          B6     1          A5     3          D4     0
 C          5          B7     1          A6     3          D5     0
 H          8          B8     5          A7     1          D6     0
 H          8          B9     5          A8     1          D7     0
 C          8          B10    5          A9     1          D8     0
 H          11         B11    8          A10    5          D9     0
 H          11         B12    8          A11    5          D10    0
 C          11         B13    8          A12    5          D11    0
 H          14         B14    11         A13    8          D12    0
 H          14         B15    11         A14    8          D13    0
 H          14         B16    11         A15    8          D14    0
 C          11         B17    8          A16    5          D15    0
 H          18         B18    11         A17    8          D16    0
 H          18         B19    11         A18    8          D17    0
 H          18         B20    11         A19    8          D18    0
 H          18         B21    11         A20    8          D19    0

   B1              1.07000000
   B2              1.07000000
   B3              1.07000000
   B4              1.54000000
   B5              1.07000000
   B6              1.07000000
   B7              1.54000000
   B8              1.07000000
   B9              1.07000000
   B10             1.54000000
   B11             1.07000000
   B12             1.07000000
   B13             1.54000000
   B14             1.07000000
   B15             1.07000000
```

```
B16              1.07000000
B17              1.90715663
B18              1.07000000
B19              1.07000000
B20              1.07000000
B21              1.07000000
A1             109.47122063
A2             109.47122063
A3             109.47122063
A4             109.47122063
A5             109.47122063
A6             109.47122063
A7             109.47122063
A8             109.47122063
A9             109.47122063
A10            109.47122063
A11            109.47122063
A12            109.47122063
A13            109.47122063
A14            109.47122063
A15            109.47122063
A16            108.42539314
A17             82.77736425
A18            108.42283070
A19            132.78888874
A20             29.58769036
D1            -120.00000000
D2            -120.00000000
D3              60.00000000
D4             -60.00000000
D5            -180.00000000
D6              60.00000000
D7             -60.00000000
D8            -180.00000000
D9              60.00000000
D10            -60.00000000
D11           -180.00000000
D12             60.00000000
D13            180.00000000
D14            -60.00000000
D15            -28.58883891
D16            -71.82906644
D17            179.99502606
D18             38.08086288
D19             82.81926042

1 2 1.0 3 1.0 4 1.0 5 1.0
```

```
2
3
4
5 6 1.0 7 1.0 8 1.0
6
7
8 9 1.0 10 1.0 11 1.0
9
10
11 12 1.0 13 1.0 14 1.0
12
13
14 15 1.0 16 1.0 17 1.0
15
16
17
18 19 1.0 20 1.0 21 1.0 22 1.0
19
20
21
22
```

Appendix B

Sequential I/O Tests Raw Data

B.1 Internal Hard Disk Sequential I/O Operation on Windows

Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
Unencrypt	55.692s	51.682s	2m11.352s	2m23.411s
	51.573s	52.291s	2m13.567s	1m53.256s
	51.807s	52.915s	2m26.874s	1m49.200s
Bitlocker	56.331s	51.308s	2m31.757s	2m18.965s
	55.380s	51.698s	2m6.890s	1m51.196s
	56.050s	51.292s	2m14.019s	1m50.838s
TrueCrypt	50.388s	52.057s	2m15.283s	2m31.101s
	50.793s	51.417s	2m6.797s	2m2.132s
	50.824s	51.558s	2m14.706s	2m0.635s
EFS	1m2.696s	1m3.304s	24m42.767s	47m36.895s
	57.985s	1m2.821s	24m45.169s	45m18.460s
	58.141s	1m3.117s	24m55.527s	47m15.382s

B.2 Internal Hard Disk eCryptfs Encryption Sequential I/O Operation

eCryptfs Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
Unencrypt	1m58.112s	2m7.627s	2m54.201s	5m40.319s
	2m1.710s	2m8.871s	3m20.282s	6m7.547s
	1m58.541s	2m10.129s	3m12.641s	5m57.726s
Unencrypt+LV	2m5.732s	2m8.813s	2m52.579s	5m29.302s
	2m4.074s	2m11.714s	2m57.283s	5m34.784s
	2m3.348s	2m10.031s	3m1.179s	5m51.252s
eCryptfs	2m39.263s	2m15.385s	6m25.790s	10m32.324s
	2m35.170s	2m12.141s	6m3.706s	11m1.453s
	2m50.511s	2m10.470s	6m15.019s	10m37.791s
eCryptfs+LV	2m43.206s	2m10.371s	6m25.790s	10m5.342s
	2m35.516s	2m13.242s	6m3.706s	10m32.324s
	2m38.749s	2m11.039s	6m15.019s	11m1.453s

B.3 Internal Hard Disk Dm-crypt Encryption Sequential I/O Operation

Dm-crypt Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
Unencrypt	2m1.892s	2m5.342s	2m48.053s	4m47.274s
Unencrypt+LV	1m59.794s	2m6.471s	3m6.895s	5m40.261s
Dmcrypt+Luks	2m0.849s	2m13.204s	2m54.493s	5m35.017s
Dmcrypt+LV	1m57.230s	2m19.112s	2m56.061s	5m38.749s
Dmcrypt+LV+Luks	1m59.885s	2m12.117s	2m49.062s	5m18.144s

B.4 External USB Hard Disk Sequential I/O Operation on Linux

Extra USB Harddisk Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
Unencrypt	1m35.173s	1m49.494s	1m22.490s	5m8.616s
	1m33.224s	1m48.956s	1m18.300s	5m6.492s
	1m32.933s	1m49.494s	1m19.678s	5m1.539s
TrueCrypt	2m4.941s	2m15.689s	1m40.915s	5m23.539s
	2m4.905s	2m16.248s	1m39.808s	5m22.599s
	2m4.923s	2m15.969s	1m42.966s	5m23.359s
Dm-crypt +Luks	2m8.000s	2m7.721s	2m9.630s	5m29.437s
	1m59.907s	2m9.803s	2m0.464s	5m25.684s
	2m5.941s	2m16.264s	2m13.047s	5m28.665s
eCryptfs	2m0.989s	2m4.114s	3m27.711s	11m5.122s
	2m24.718s	2m0.447s	3m21.792s	11m1.806s
	2m3.553s	1m59.256s	3m23.750s	11m8.236s

B.5 Internal Hard Disk Sequential I/O Operation on Linux

Internal Harddisk Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
Unencrypt	2m0.957s	2m3.966s	2m43.223s	4m48.438s
	2m5.252s	2m3.403s	3m7.009s	5m24.349s
	2m3.227s	2m4.155s	3m1.203s	5m10.893s
Dm-crypt	2m1.990s	2m7.291s	2m49.988s	5m17.442s
	1m59.689s	2m13.428s	3m6.769s	5m56.672s
	1m58.607s	2m15.043s	2m52.159s	5m35.208s
eCryptfs	2m47.234s	2m9.249s	6m51.898s	8m44.146s
	2m48.873s	2m17.736s	5m44.329s	7m41.721s
	2m48.566s	2m12.921s	6m4.377s	8m34.028s

B.6 TrueCrypt Encryption Sequential I/O Operation on Windows and Linux

TrueCrypt Sequential I/O (Real time)	3G File Write	3G File Read	2.3G Directory Write	2.3G Directory Read
On Windows 7	8m49.839s	4m38.398s	19m53.402s	5m17.694s
	8m51.726s	4m38.803s	20m36.738s	5m13.466s
	8m50.619s	4m48.896s	21m23.851s	5m10.206s
On Redhat 5.4	7m42.351s	5m9.733s	6m14.180s	7m3.917s
	7m14.205s	5m11.245s	5m49.568s	7m40.190s
	7m51.380s	5m9.019s	5m59.670s	7m36.829s

Appendix C

Bonnie++ Benchmark Results Raw Data

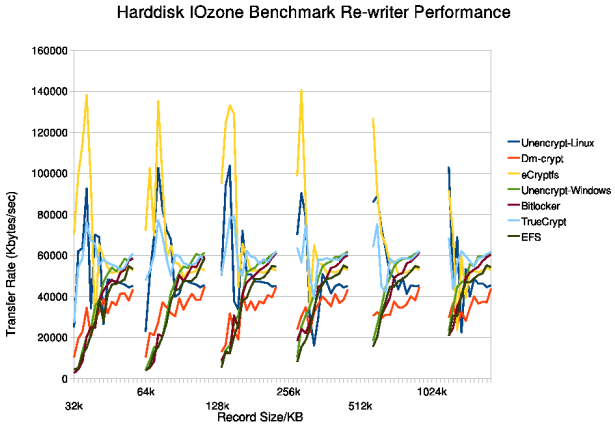
Bonnie++	Sequential Block Operations					
	Write		Re-write		Read	
	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU
Internal Harddisk						
Unencrypt	58295	13	24358	0	54584	0
Dm-crypt	51060	12	23922	1	57163	0
eCryptfs	36854	92	21570	49	56951	47
Extra USB Harddisk						
Unencrypt	33024	7	12683	0	29985	0
Dm-crypt	28114	6	11084	0	32578	0
eCryptfs	27082	68	12252	21	29005	12
Small USB Stick						
Unencrypt	4866	1	2606	0	6458	0
Dm-crypt	4587	1	3442	0	11808	0
eCryptfs	4173	10	3441	0	10160	0

Bonnie++	Random Seek	Sequential File Create		Random File Create	
	/sec	K/sec	%CPU	K/sec	%CPU
Internal Harddisk					
Unencrypt	177.4	15903	94	14206	83
Dm-crypt	163.7	16158	97	16006	95
eCryptfs	130.9	2787	90	2832	91
Extra USB Harddisk					
Unencrypt	163.2	18233	95	18454	98
Dm-crypt	163.3	18005	94	18372	96
eCryptfs	95.8	2781	89	2929	96
Small USB Stick					
Unencrypt	52.8	13781	70	13849	69
Dm-crypt	33.1	13853	71	13829	68
eCryptfs	25.9	1308	56	1986	64

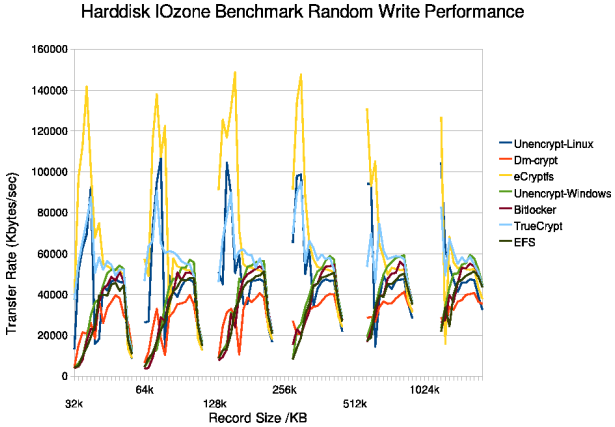
Appendix D

IOzone Benchmark Results

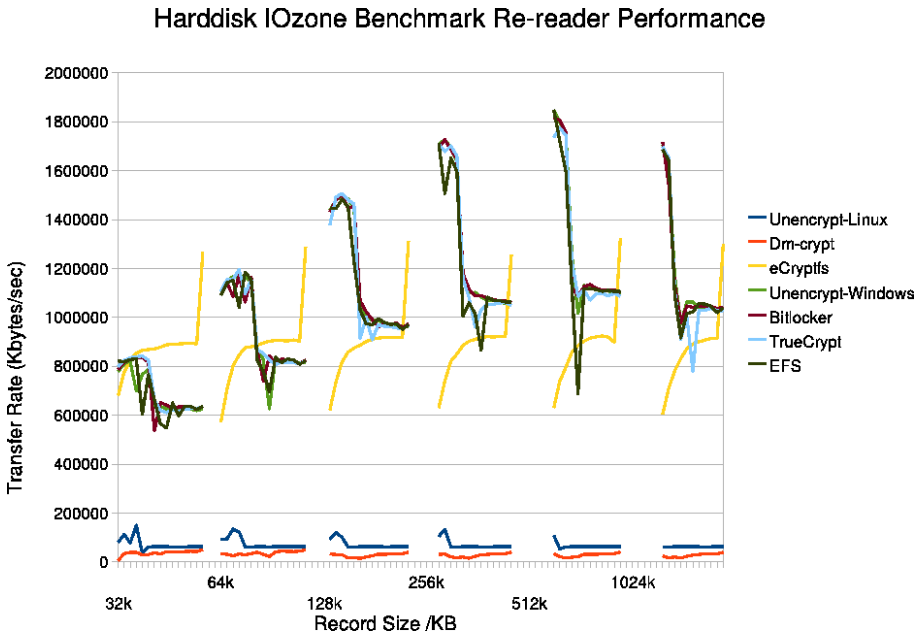
D.1 Different Tools IOzone Benchmark Performace Rewriter Report



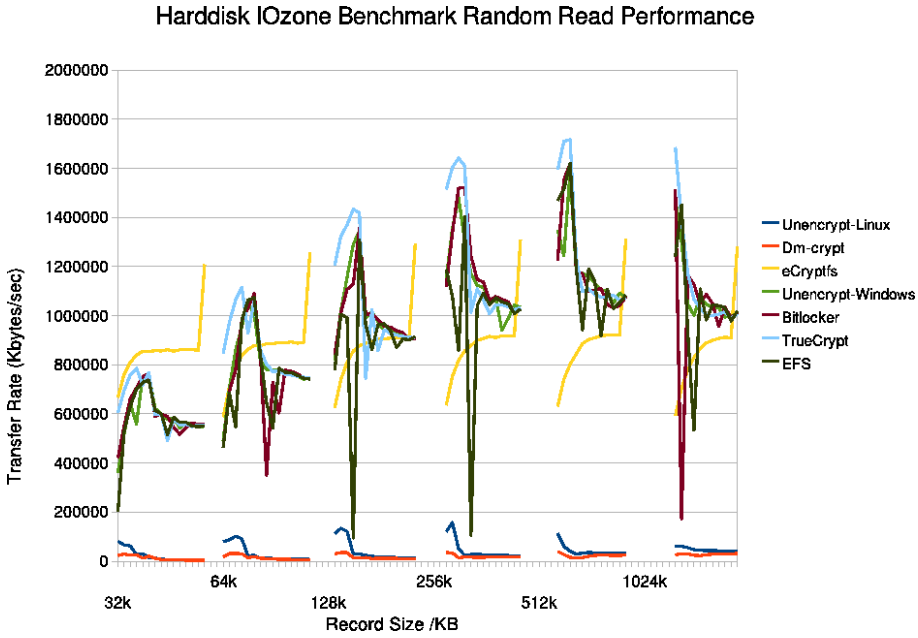
D.2 Different Tools IOzone Benchmark Performace Random Write Report



D.3 Different Tools IOzone Benchmark Performace Rereader Report



D.4 Different Tools IOzone Benchmark Performace Random Read Report



D.5 IOzone Benchmark Tests Raw Data

Unencrypted Internal Hard Disk (Windows)

KB	reclen	write	rewrite	read	reread	rand read	rand write	rec rewrite
64	32	3832	4347	661587	778743	361156	3880	4706
64	64	3862	3941	1045585	1107546	491837	4527	3858
128	32	5107	4694	786940	810399	520065	6254	7951
128	64	5750	9188	1107546	1145737	706944	7391	10336
128	128	9135	7859	1329055	1437679	819280	8798	7930
256	32	10274	13444	746660	821531	641711	9338	15940
256	64	15300	15303	1107546	1168114	858070	11418	15914
256	128	13636	13633	1427386	1480383	1020605	12508	15658
256	256	8445	10338	1612061	1708785	1147936	8315	10401
512	32	20249	20188	819842	698685	558948	18438	31439
512	64	13308	15797	944080	1160184	956919	13051	20986
512	128	16236	16236	1455170	1497058	1156817	15879	31988
512	256	25383	25382	1663629	1726045	1354642	22825	25038
512	512	20577	18175	1652140	1848763	1350056	19033	21129
1024	32	25395	24148	813707	766762	752532	29652	59169
1024	64	21016	20456	1127911	1170400	1052022	23208	59086
1024	128	30650	26754	1424835	1484977	1290344	31287	65486
1024	256	23476	30668	1627414	1693064	1479467	25700	45701
1024	512	24546	26408	1564617	1797370	1244045	25237	23298
1024	1024	21799	23329	1427386	1706347	1475817	23751	23428
2048	32	30765	35437	795048	787070	759580	36158	84357
2048	64	30691	30687	1093372	1141909	1008345	34813	161630
2048	128	27454	30907	1401053	1443753	1345500	28778	67225
2048	256	35048	35077	1580118	1635760	1304416	33862	110913
2048	512	33432	35077	1626308	1753242	1584304	33667	80023
2048	1024	35067	35075	1474907	1598596	1299103	34053	50037
4096	32	40947	37975	640294	637479	621780	37611	153143
4096	64	40935	41985	835081	867639	816484	43549	169597
4096	128	41008	40946	1047416	1056902	993067	47659	254074
4096	256	40066	40943	1120121	1181820	1171403	39446	100233
4096	512	44685	44681	1204733	1256624	1181382	42619	164628
4096	1024	43751	44651	974757	1166833	1047531	43411	96496
8192	32	42350	47012	627673	635996	600967	44694	231198
8192	64	46853	46579	780744	827964	780649	47597	302191
8192	128	49198	51779	980551	998559	1018216	47343	312473
8192	256	46848	49193	1071878	1101110	1124597	46731	258159
8192	512	46851	49197	826819	1017188	1172911	49804	189708
8192	1024	46851	46854	881750	975453	1001957	50088	192853
16384	32	47551	51010	628312	626113	578610	50379	332870
16384	64	50374	51750	816310	626184	780712	47932	405131
16384	128	51751	53148	973567	987864	974187	53600	444562
16384	256	53157	54624	1078675	1101997	1113765	51270	220668
16384	512	51523	54653	1102315	1130075	1159586	53842	448395
16384	1024	49439	48232	1030358	1062860	1076006	50233	333389
32768	32	53849	54149	617367	633260	583434	52174	424138
32768	64	54614	54631	813128	828995	778125	53101	637505
32768	128	53151	54146	969658	988247	937478	54199	710678
32768	256	55401	56207	1061430	1082320	1062211	52660	815875
32768	512	57001	57007	1104940	1127412	1131729	55722	757146
32768	1024	55401	55859	1034549	1062904	1045771	54609	498045

65536	32	53124	52305	622313	623321	543360	52063	438620
65536	64	53869	53870	814062	821597	771293	53059	663784
65536	128	55804	56608	956830	974292	941080	55765	890447
65536	256	57010	57011	1048765	1071496	1071601	56791	928152
65536	512	57255	58278	1087864	1112172	1106977	57984	921621
65536	1024	55126	54739	1022145	1042516	1036104	55357	698449
131072	32	54451	54841	622340	629232	564492	54166	493167
131072	64	56731	57186	811153	824455	764522	56913	790597
131072	128	58705	56200	957787	969981	932165	563371040003	
131072	256	58875	59287	1053194	1069501	940814	588661162531	
131072	512	58030	58496	1084958	1109062	1098995	584821254339	
131072	1024	60751	59608	1022121	1038588	1030781	593921141517	
262144	32	55117	58411	620455	625889	558941	52287	522082
262144	64	57220	60758	809221	819377	755286	55827	784304
262144	128	58131	58696	953018	968940	922139	563011075625	
262144	256	59132	59157	1036196	1064049	988671	570211342621	
262144	512	59050	59363	1085058	1107672	1050141	591371367037	
262144	1024	58821	59494	1021973	1037134	994536	577221463001	
524288	32	56569	57330	619131	618612	546198	22025	533447
524288	64	58672	59866	805942	814959	740716	26634	811658
524288	128	59200	60313	950198	962758	905435	329561130724	
524288	256	59779	60674	1042063	1057168	1043067	382871416362	
524288	512	60000	60815	1079398	1098115	1091862	447771535353	
524288	1024	60116	59738	1015366	1031529	997365	524271665686	
1048576	32	59634	58555	67965	625528	549741	12884	536451
1048576	64	58965	61247	63331	815806	747180	16800	838625
1048576	128	60179	61619	63395	961075	910726	220011165685	
1048576	256	59943	61758	63296	1055314	1034064	283481448947	
1048576	512	60698	61710	1091709	1102599	1073279	363841658722	
1048576	1024	60288	61849	1019336	1037300	1011192	445911734236	

Internal Hard Disk Bitlocker Encryption (Windows)

KB	reclen	write	rewrite	read	reread	rand read	rand write	rec rewrite
64	32	2710	2708	725815	786937	421177	3993	2356
64	64	2906	3949	1052944	1091372	465788	3614	2991
128	32	3694	4356	793199	817038	549698	4792	4668
128	64	5307	5307	1119985	1141359	692213	4095	7544
128	128	4787	8892	1390865	1430795	839989	7785	8984
256	32	8126	8127	818156	828355	662317	7887	9656
256	64	8194	8235	983671	1085430	779755	9064	14058
256	128	12283	12432	1427380	1480377	1011966	9449	14083
256	256	9450	18317	1625196	1694255	1117892	15422	18467
512	32	15333	20423	820400	835880	705690	17257	31233
512	64	15924	21501	1113728	1171542	978841	16982	28296
512	128	16374	12271	1429085	1502694	1107541	13222	32608
512	256	16634	24150	1638554	1726038	1337969	22973	25761
512	512	20798	15486	1432508	1815090	1224303	16767	15760
1024	32	20521	24879	813703	836172	752528	23146	63326
1024	64	24544	20458	1137019	1062767	1032048	28864	58392
1024	128	20455	30658	1319519	1446365	1127906	30455	52209
1024	256	22043	22081	1600193	1685898	1518913	20536	40240

1024	512	19784	20459	1626301	1806865	1553434	22196	31312
1024	1024	25003	21017	1456936	1717364	1516026	26265	25829
2048	32	22333	24769	797429	817736	763943	22667	136499
2048	64	24606	27345	1132176	1162716	1089881	26933	85977
2048	128	22063	24649	1392079	1454279	1358097	23986	90711
2048	256	24608	27346	1572322	1647014	1522781	26392	82190
2048	512	24610	30769	1607181	1756453	1619694	29512	50036
2048	1024	30696	24525	1362739	1542914	173991	27531	54000
4096	32	32759	38074	646739	536477	590433	35919	137806
4096	64	35145	37852	828211	864656	846004	37757	155194
4096	128	35147	41007	950358	1067986	1003792	39978	196600
4096	256	30883	32792	1133115	1180211	1243716	31702	130620
4096	512	32722	35091	1155276	1121823	1184301	39501	115975
4096	1024	32767	35146	1012073	1131507	1164272	39640	85007
8192	32	36404	42995	644019	650918	599871	42675	177108
8192	64	39199	42740	704754	739045	349762	41062	297993
8192	128	44916	46846	978891	1023384	1009511	47255	271188
8192	256	39323	44721	1095746	1115351	1147656	44446	235296
8192	512	37821	39349	1094931	1087095	1171757	39170	209597
8192	1024	41456	39788	1053233	977691	1128039	36364	193862
16384	32	43881	45810	629736	641859	589451	44489	391632
16384	64	49156	50423	813980	841966	729287	50762	441147
16384	128	47956	48844	968587	991211	982913	49840	417912
16384	256	43721	45752	1044696	1088982	1137560	45387	401390
16384	512	45732	49185	1106165	1125849	1098072	47740	310613
16384	1024	46827	49186	1032995	1046581	1048962	45525	290293
32768	32	48537	48216	625910	628805	545534	48612	397153
32768	64	47052	47391	721601	813937	604061	46860	570475
32768	128	52446	50437	957466	961253	956867	51151	689755
32768	256	49789	49157	1059355	1089446	1061308	50672	745471
32768	512	49680	50631	1070106	1134424	1103772	50221	617325
32768	1024	51755	51506	1028665	1041455	1088068	53069	574706
65536	32	46013	46410	627143	634396	516544	46849	465078
65536	64	51130	50758	817138	830214	778898	50591	721234
65536	128	52449	52802	960957	975834	953411	52592	922178
65536	256	53885	53891	1056482	1078754	1077903	53632	931327
65536	512	50764	51427	1085830	1119314	1111190	50630	888722
65536	1024	52807	53165	1014709	1044874	1049249	52823	904719
131072	32	50893	51642	628548	634385	543643	50936	507396
131072	64	50471	54700	811265	824994	772937	50780	788629
131072	128	54029	54077	956270	970841	938701	537601015180	
131072	256	54069	54579	1050790	1073629	1066907	538791219000	
131072	512	55795	56198	1078355	1111965	1045784	559961195019	
131072	1024	53766	54447	1030014	1045074	956094	550341118533	
262144	32	52331	52983	625188	631341	562916	46567	526952
262144	64	55002	54235	814652	824494	763846	49430	786197
262144	128	55889	55771	959919	977612	931513	519621122905	
262144	256	56754	56283	1050861	1067401	1056484	545791279456	
262144	512	56452	57015	1084473	1112809	1031751	541151413569	
262144	1024	56293	57327	1020024	1043935	1035817	538051420157	
524288	32	56153	57619	622955	624306	558143	20166	530268
524288	64	56300	58227	810812	810371	745462	26129	814094
524288	128	57385	58880	953405	957226	903689	328351139531	
524288	256	57008	58936	1050286	1067648	1008382	370491409703	
524288	512	58664	59105	1088622	1112554	1041305	442351578126	
524288	1024	58097	59137	1023547	1036807	990677	506411571715	
1048576	32	58160	58568	625955	635849	558522	11741	543632

1048576	64	57462	59521	810217	825794	745557	15899	845635
1048576	128	59361	61138	955414	973596	908251	218331	186517
1048576	256	58442	60004	1050789	1059458	1030742	276141	426666
1048576	512	60079	61427	1092287	1100621	1078074	360691	641147
1048576	1024	58741	60407	1017137	1042177	1013632	436131	711984

Internal Hard Disk EFS Encryption (Windows)

KB	reclen	write	rewrite	read	reread	rand read	rand write	record rewrite
64	32	3837	4311	725822	821534	201508	4445	3587
64	64	4754	3837	1003485	1091382	462908	4488	3867
128	32	4285	5137	766765	819284	528336	5026	5880
128	64	9115	5721	1107550	1141369	685868	8721	12476
128	128	7629	5218	1390877	1444631	778746	8811	7859
256	32	8590	11740	803867	827216	632217	9951	15294
256	64	12682	12660	1060420	1152365	546688	11364	11798
256	128	13237	13235	1410559	1444631	1005172	12739	15061
256	256	7462	8055	1620806	1713688	1189020	8222	7812
512	32	14815	14815	783336	830663	700324	13962	28959
512	64	20433	15368	1164707	1041946	981259	16912	28460
512	128	12522	12314	1403937	1482224	991836	13385	30794
512	256	15428	15402	1491464	1506492	1079562	13988	26296
512	512	14865	15596	1638568	1848771	1469477	17413	15062
1024	32	19903	21206	830086	603661	728474	19187	71007
1024	64	20396	20394	1139195	1184898	1063721	22585	55767
1024	128	20724	20474	1334994	1452525	94944	20652	59819
1024	256	18690	18637	1568727	1652147	858381	19043	52610
1024	512	20376	20355	1339479	1721086	1514120	18961	31164
1024	1024	20931	20756	1320259	1687101	1241468	21859	21225
2048	32	24596	27551	807530	765784	737229	24990	84868
2048	64	24574	27344	1116076	1154589	1073506	26759	130080
2048	128	27383	26578	373360	1227769	1308345	33229	75625
2048	256	31539	24302	1545919	1595937	1403937	30051	91708
2048	512	27780	30773	1535007	1597535	1621355	33620	72860
2048	1024	32833	30763	1343616	1641378	1450763	33821	46750
4096	32	37243	38618	631591	661361	614003	39453	149797
4096	64	35126	37798	851962	824507	839996	35132	116802
4096	128	35114	37799	1006334	1029836	963184	37008	215024
4096	256	35145	34355	1069786	1005066	106543	35520	106179
4096	512	35970	37799	1138111	1119602	1192131	39371	108935
4096	1024	38898	30407	924474	1089518	933584	24498	65724
8192	32	36603	27730	575368	564873	600648	39892	253991
8192	64	35094	42776	805932	778081	644132	41586	274520
8192	128	40993	39371	990758	973621	862287	43456	215179
8192	256	23422	42065	912398	1059716	1040926	43891	174390
8192	512	41804	44682	799635	688830	942921	38870	162861
8192	1024	40054	40989	424384	916198	534863	40118	150395
16384	32	42190	41087	570250	547831	516356	39507	327534
16384	64	43069	43716	371556	694505	542781	43766	499588
16384	128	45913	44694	980379	970019	963572	46179	458148
16384	256	44714	45710	945460	1015115	1091755	45457	448396
16384	512	43704	44865	1110668	1116857	1189464	45042	402208

16384	1024	46386	47101	993458	1016787	1109026	45947	244618
32768	32	44714	45784	633556	650990	586359	44848	441011
32768	64	46840	47401	813710	837286	785748	47545	571327
32768	128	45843	48573	970843	994697	966785	47026	625118
32768	256	47586	48573	1018667	865661	1044662	48222	672286
32768	512	47505	47377	893048	1116190	1138619	45219	691994
32768	1024	47417	46832	724379	1021536	982796	47223	439985
65536	32	45489	46133	601791	597144	565407	45673	476443
65536	64	46709	47687	807296	815131	769958	46682	687732
65536	128	47849	48422	952512	979821	938776	48863	861690
65536	256	48546	48873	1052323	1082217	1073123	48843	835053
65536	512	47746	48519	1084079	1114346	917318	47046	952820
65536	1024	49179	48870	1040622	1056943	1043551	49133	790454
131072	32	48121	47122	628007	635546	566522	41833	485348
131072	64	48272	47686	811960	830124	768220	48130	727764
131072	128	48170	49340	951524	971148	870457	49338	940855
131072	256	49814	50116	1049765	1069532	1055602	497471143355	
131072	512	49822	50116	1092336	1106470	1107066	493421180317	
131072	1024	50117	50436	1023181	1054526	1041149	502151011681	
262144	32	47498	48273	627712	635733	549729	44526	508046
262144	64	55643	49559	813209	825675	756783	48177	812328
262144	128	49593	51608	960337	973510	905473	511801086356	
262144	256	51521	51174	1050178	1068763	1048996	506561363846	
262144	512	51207	52192	1087255	1110593	1029536	502001388585	
262144	1024	51723	51257	1030262	1045341	1032722	513761313934	
524288	32	50235	54768	624714	624377	548491	17794	513040
524288	64	53845	54544	812756	808616	745018	24877	823320
524288	128	51737	54925	955750	951354	901660	297761126160	
524288	256	54256	55069	1045845	1062558	1008766	360931430750	
524288	512	52149	54829	1082555	1105859	1052748	422751512966	
524288	1024	51935	55263	1022486	1019659	978295	487971575333	
1048576	32	52007	53386	52712	635468	550771	10850	522639
1048576	64	52084	58639	54470	821985	742060	14835	829965
1048576	128	53149	54521	52453	969388	915262	207661156563	
1048576	256	53510	54029	50191	1063021	1026929	268131465542	
1048576	512	52419	54192	47609	1106757	1083330	351061622216	
1048576	1024	53487	54216	1014763	1038083	1019351	435311703374	

Internal Hard Disk TrueCrypt (Windows)

KB	reclen	write	rewrite	read	reread	rand read	rand write	record rewrite
64	32	32003	27131	729362	799568	605341	37435	50377
64	64	47211	47815	1075678	1107550	844742	46579	45419
128	32	55542	54193	791107	823798	695438	55326	103045
128	64	52115	52684	1128447	1154589	974067	63679	86452
128	128	59700	50624	1397377	1378058	1205801	46155	61404
256	32	61473	58663	812605	835303	758981	65903	114508
256	64	63740	59819	1161315	1156822	1064194	66166	155952
256	128	58235	63048	1455176	1491464	1323180	67267	104558
256	256	71600	63828	1647596	1708792	1514120	69795	63645
512	32	78964	75931	817045	838818	785909	75984	257958
512	64	64385	77241	1153476	1194959	1115816	90645	251451

512	128	85178	78118	1432520	1506492	1370165	78328	199959
512	256	83483	56652	1578040	1677636	1603424	89720	123276
512	512	84617	63948	1616425	1736073	1594873	53576	50128
1024	32	81210	67406	765539	842066	718411	87591	328930
1024	64	99200	69388	1151810	1095379	929774	64969	389183
1024	128	76992	79037	1433379	1480389	1434238	89926	229544
1024	256	83949	74508	1607735	1701500	1641942	94955	194988
1024	512	106495	75196	1604500	1774710	1710014	69862	179832
1024	1024	83501	68748	1422300	1697877	1685912	82801	78085
2048	32	81919	65521	806849	825646	768613	50504	414611
2048	64	82843	58238	1072303	1156543	1058075	60252	541552
2048	128	72044	53501	1406413	1463633	1418085	57574	182905
2048	256	86696	49903	1565135	1659597	1609357	55520	393601
2048	512	58795	44784	1615334	1744301	1717379	45969	230639
2048	1024	54062	43451	1414312	1652147	1390069	49370	65929
4096	32	58818	55283	605763	661132	596660	58084	436572
4096	64	65882	50226	814611	868273	860156	61184	574073
4096	128	57258	50140	972286	914316	744803	61270	671855
4096	256	62573	39120	1129913	1156543	1014120	66217	591204
4096	512	68655	42293	1085070	1228558	1233785	74561	433802
4096	1024	53069	56053	835303	1106654	1173563	67266	247619
8192	32	59869	58159	636697	619849	600328	52104	509476
8192	64	57990	60289	827717	852759	802620	60748	700632
8192	128	62139	62252	992247	1000652	1023886	62129	833375
8192	256	62128	57942	1050583	1080537	1108834	62829	802552
8192	512	63118	58539	1078649	1086055	1098334	61557	708937
8192	1024	62754	62055	675245	911095	1067280	60819	471902
16384	32	55288	56387	613963	610799	491057	56345	507524
16384	64	57535	59721	820460	826341	773224	58978	732867
16384	128	60395	59548	932583	907768	857996	59547	992633
16384	256	57492	57890	919898	961346	1063071	568771044876	
16384	512	57245	56259	1062923	1112961	1108898	57427	936553
16384	1024	57533	59672	974464	1003485	1030695	54990	745021
32768	32	54754	55514	618507	624364	578791	54456	528132
32768	64	54425	56342	795795	807173	774845	56276	828998
32768	128	58125	58873	948693	969748	942666	592821110329	
32768	256	58599	58756	1009573	1030834	1008110	595711068472	
32768	512	57695	57934	1055057	1071058	1086040	581511367375	
32768	1024	58341	57127	940327	781146	1002605	581941102526	
65536	32	54473	54738	607409	615563	554565	49235	518999
65536	64	55059	55463	798376	817481	760538	55090	835882
65536	128	55225	55880	945758	963530	928168	530961149044	
65536	256	57815	57918	1033311	1053888	1044669	573741386519	
65536	512	57168	58776	1068763	1097067	1076435	587571499601	
65536	1024	55692	54691	984135	1029670	1001313	557121421072	
131072	32	52341	52881	616777	625279	557706	52309	539608
131072	64	54608	55458	796078	817063	758346	54648	851872
131072	128	57629	58139	939747	962164	920857	577331153558	
131072	256	57936	58108	1020065	1053329	1044145	579971450145	
131072	512	58266	58529	1060769	1097842	1075875	585111603567	
131072	1024	58149	58563	1004713	1028895	1006123	584631622162	
262144	32	55111	54087	618968	627095	548839	52151	544801
262144	64	56066	56439	804011	814059	752997	49092	856393
262144	128	56586	56834	944550	956822	915356	517541202460	
262144	256	59773	57868	1028482	1058613	1037027	521041493254	
262144	512	58288	58119	1076049	1088300	1083425	573721668072	
262144	1024	58221	59171	1010667	1036959	1016215	546191747586	

524288	32	56823	58261	624590	627114	555288	20415	548593
524288	64	58962	59838	811474	810828	749611	25691	862522
524288	128	58329	59602	952860	953820	913253	321071	210157
524288	256	59836	60473	1043044	1060423	1023863	370571	508527
524288	512	59107	60066	1080467	1098123	1063279	439361	707320
524288	1024	59458	60707	1015328	1028593	989376	510461	796239
1048576	32	59308	60983	62930	634852	552105	11794	543928
1048576	64	59313	57437	807288	821605	746342	15988	858896
1048576	128	59948	62085	948429	967287	908374	215351	213323
1048576	256	59816	60760	1042489	1048559	1032845	277091	501319
1048576	512	60677	62182	1081915	1086317	1073073	359581	720001
1048576	1024	59871	61753	1015656	1024027	1007005	438901	829722

Unencrypted Internal Hard Disk (Linux)

KB	reclen	write	rewrite	read	reread	rand read	rand write	record rewrite
64	32	19030	24883	145	79406	81632	13341	43095
64	64	23694	22832	66593	92083	79800	26272	27349
128	32	29871	62013	124023	113264	66252	51159	106932
128	64	30755	50451	99300	94532	87252	26823	63747
128	128	34990	50257	97935	92353	112379	50914	48966
256	32	54527	63540	131747	76623	62500	62515	138899
256	64	44329	69378	129159	134179	101870	72051	104748
256	128	53601	93841	109965	120929	134171	45054	53735
256	256	54759	70004	111358	104065	120470	65339	52213
512	32	74353	92601	151609	149925	27767	69395	249755
512	64	60477	102685	147633	122751	92003	94604	160551
512	128	76361	103772	126952	100296	121298	104425	138228
512	256	65616	90315	132918	133370	157684	97971	144794
512	512	68294	85991	98576	109005	113424	94169	100115
1024	32	89919	34150	13357	37082	30292	92277	370063
1024	64	35404	82633	61988	61850	22130	106733	319594
1024	128	78030	37626	61858	61783	29260	89268	256192
1024	256	71990	78701	61757	61843	51431	98736	210746
1024	512	40488	88965	61809	53111	60624	94152	128368
1024	1024	14824	103329	60941	60909	60313	104543	88934
2048	32	51122	70189	61731	61714	15077	15767	377237
2048	64	16693	72318	61779	61714	24044	13013	335193
2048	128	32986	33749	61619	61623	30586	50489	385250
2048	256	9639	29555	48122	61636	23525	50112	252592
2048	512	16849	73339	61638	61662	40775	14462	236678
2048	1024	28704	46118	61564	61625	61045	61710	92140
4096	32	36745	68920	56741	62776	12436	18188	572218
4096	64	14293	67737	56925	62779	12453	49378	543372
4096	128	16829	72050	62733	62710	23864	59201	577795
4096	256	50258	16156	62753	62755	27507	61717	483361
4096	512	15565	62870	57591	62730	29142	37348	388508
4096	1024	24410	68891	57246	62749	55403	53588	342560
8192	32	54388	26555	61749	62652	8774	44400	640900
8192	64	29919	39915	58375	62650	14994	41160	638204
8192	128	24288	50773	60795	62616	20609	35479	631902
8192	256	34369	30061	59164	62628	29975	34315	554864

8192	512	26030	51084	62618	62630	33091	49358	563409
8192	1024	53415	22394	61627	62622	47077	47280	456634
16384	32	33482	48140	61366	62919	6526	41915	701280
16384	64	32306	41412	60922	61933	11278	38946	605156
16384	128	34056	50859	62086	62901	17342	35430	601864
16384	256	41128	50929	61588	62912	25873	42804	681556
16384	512	32764	48986	60759	62900	35743	44779	651760
16384	1024	35424	53349	61634	62907	44663	41399	578325
32768	32	47645	47898	62305	61778	5765	46283	658839
32768	64	43459	48079	62137	62898	10170	44047	728160
32768	128	43247	47317	62656	62891	17105	46865	706923
32768	256	47300	47124	62429	61861	25777	46224	689199
32768	512	44470	43570	62237	62892	34643	42795	712364
32768	1024	44641	40019	62049	62894	43463	45781	651179
65536	32	47712	47183	62216	62422	5280	47465	700800
65536	64	47611	47320	62110	62553	9706	47180	734008
65536	128	48215	47297	62507	62274	15843	46898	514080
65536	256	47472	41501	62387	62136	24459	47462	756102
65536	512	48837	49005	62275	62472	34320	46895	744811
65536	1024	48026	48527	62134	62330	44178	45803	720143
131072	32	47705	46580	62262	62270	4973	46079	732822
131072	64	48316	46409	62431	62435	8961	47325	739071
131072	128	48351	46871	62468	62342	15345	47468	761528
131072	256	47716	45055	62297	62419	24549	46624	766615
131072	512	48243	41345	62325	62336	33803	47182	766170
131072	1024	48374	46239	62382	62269	42731	47466	730979
262144	32	50158	45820	62514	62519	4612	45899	739734
262144	64	50226	45795	62444	62455	8370	45845	756572
262144	128	50295	46451	62496	62505	14457	46247	769612
262144	256	50370	46067	62528	62475	23591	46869	773525
262144	512	50292	45463	62475	62539	32558	47577	769371
262144	1024	50280	46401	62462	62413	42672	47008	770748
524288	32	52062	44389	62578	62541	4315	14506	736411
524288	64	52303	44450	62569	62533	7920	19354	757291
524288	128	52319	44890	62575	62509	13755	24003	773888
524288	256	52324	44365	62541	62564	22315	30964	775837
524288	512	52238	44936	62583	62573	32062	35485	779901
524288	1024	52143	44543	61473	62569	41626	39251	779261
1048576	32	53078	45267	62447	62485	4045	8635	744156
1048576	64	53233	45550	62448	62447	7499	12792	762100
1048576	128	53311	45011	61683	62493	13156	16763	768148
1048576	256	53290	45136	62487	62447	21507	21892	779805
1048576	512	53294	45273	62447	62458	31368	28255	779958
1048576	1024	53288	45449	62448	62494	40481	32451	785763

Internal Hard Disk Dm-crypt (Linux)

KB	reclen	write	rewrite	read	reread	rand read	rand write	record rewrite
64	32	4757	10344	34499	5894	23138	4320	12944
64	64	5173	10178	32802	34022	18028	6582	9864
128	32	5804	19467	30353	34178	29746	15108	24916
128	64	6253	22245	8341	31114	31403	11764	14809

128	128	592	12942	30089	32922	30338	8780	13324
256	32	9622	22725	18049	39042	24462	21498	50702
256	64	12305	21125	18811	24956	31499	23279	40998
256	128	13956	16303	33956	30238	35764	24313	38776
256	256	14319	23918	33556	30658	36132	27112	24089
512	32	19824	34408	43768	39506	25954	20777	114134
512	64	19867	27555	21392	33089	31164	32993	87014
512	128	19579	31789	29354	29967	35320	31215	47097
512	256	23755	30443	31938	32607	33936	20868	45898
512	512	17104	30532	21375	33646	40404	28611	32439
1024	32	23483	20503	30770	28482	12958	26141	191401
1024	64	11659	37085	36431	28623	14510	18275	109577
1024	128	21890	23071	9402	16314	13865	33021	49761
1024	256	27064	34725	16207	20637	14327	20421	64879
1024	512	22995	32420	8353	21460	27042	28970	55941
1024	1024	19050	29706	18795	24276	26378	28878	31991
2048	32	32830	28386	39999	30150	20302	18560	272666
2048	64	26105	34652	35999	34142	19047	10417	312866
2048	128	27674	19061	17056	17761	13522	29100	192355
2048	256	25141	22572	17733	17107	17406	27959	160427
2048	512	25772	29449	21986	16115	16108	26739	134076
2048	1024	25415	36655	17755	16336	31180	23900	67649
2048	2048	23859	28702	18215	15160	22366	23946	33857
4096	32	27320	32711	28486	37330	12108	37292	487155
4096	64	28911	31813	31023	39623	12576	29136	409964
4096	128	12894	41917	21189	14622	15373	10636	246496
4096	256	30953	32036	15142	20304	16670	32440	300559
4096	512	25633	31129	20443	19125	13968	33809	188513
4096	1024	26181	25806	14637	20329	29368	33976	111271
4096	2048	9056	30332	22597	16381	35466	24670	76326
4096	4096	31659	38996	21090	18043	29021	31821	36818
8192	32	29154	38942	31311	32944	7457	26097	379242
8192	64	27155	30301	41714	31299	10844	31260	612392
8192	128	16598	33347	21900	20039	10924	32649	474404
8192	256	30787	36277	21504	15059	15654	33924	337604
8192	512	32934	30988	21548	21538	15321	36578	320198
8192	1024	31448	32583	20531	15848	24635	32625	159052
8192	2048	30509	29761	23325	17252	29475	31369	126579
8192	4096	22767	31685	14552	23196	29105	33505	61091
8192	8192	35126	31200	22333	18866	22422	30595	39971
16384	32	36457	31855	41597	41749	5824	33546	612027
16384	64	38512	39146	42882	21530	9209	35308	550906
16384	128	32340	37602	18372	25292	11971	38545	442632
16384	256	28568	39947	24401	23075	15408	34462	459424
16384	512	37242	37547	22346	31145	20037	35493	377642
16384	1024	35337	28066	26621	23378	22819	36476	293368
16384	2048	31220	41561	25726	23911	27354	34887	192198
16384	4096	31492	34679	24380	30500	24043	35355	117966
16384	8192	37460	33144	24136	26649	31725	34264	67571
16384	16384	31640	39058	20792	26781	39448	31323	41972
32768	32	38371	37384	39965	41096	5169	36457	677725
32768	64	31830	33560	35413	41480	8191	35603	590743
32768	128	36340	33395	31426	31533	11686	36325	688459
32768	256	33662	37741	27441	29571	18353	36673	635715
32768	512	37524	34739	32298	30940	24194	35708	568879
32768	1024	36232	34413	28086	27740	26962	37311	412939
32768	2048	32750	33849	26078	27859	29287	38098	317159

32768	4096	35380	39547	28775	27896	32444	35976	209266
32768	8192	39334	39493	29535	31317	42349	34239	117817
32768	16384	37475	36670	28658	29814	42463	38900	62902
65536	32	36291	35815	44549	40803	4695	39626	604275
65536	64	38417	37912	44242	44288	7837	36456	635303
65536	128	35733	37591	32648	30962	11559	38117	632183
65536	256	36139	35286	32746	30754	18702	39111	653459
65536	512	38705	34533	32763	30654	24754	38023	566269
65536	1024	37485	39977	32424	30709	28706	39831	543546
65536	2048	40112	38167	32259	31086	30783	36024	433671
65536	4096	33331	37868	33017	31574	33683	34794	321147
65536	8192	36461	35457	32778	32041	36689	34482	225603
65536	16384	39894	40321	32302	32040	40535	37949	133133
131072	32	39600	41320	44621	42360	4517	38207	666754
131072	64	39036	41320	43991	43256	7457	39657	698272
131072	128	39860	36646	33133	32534	11552	40672	696324
131072	256	37603	41238	33380	33335	18702	40242	716283
131072	512	39410	36304	33294	33620	24539	39706	694991
131072	1024	40037	36570	33102	32817	30076	40179	648287
131072	2048	39147	37779	33361	32102	32494	38059	599145
131072	4096	40456	39936	33543	33188	34154	38433	476672
131072	8192	39125	41203	33238	33463	39086	35860	322097
131072	16384	40875	37948	33328	32942	40331	35404	197109
262144	32	41216	41484	44564	43799	4141	29661	716884
262144	64	42908	38290	42877	42406	6900	35133	730407
262144	128	41997	40664	33969	33542	10885	38326	743533
262144	256	42930	36850	34070	33756	17504	39952	728293
262144	512	42712	40104	33646	33631	23832	41276	741516
262144	1024	41576	37394	33908	33655	28850	40795	686623
262144	2048	41430	39455	33793	33055	28511	40500	658682
262144	4096	42337	36207	33909	33503	33203	38352	579781
262144	8192	42132	36930	33853	33484	37648	35450	433289
262144	16384	42647	38135	33791	33072	41206	36091	314865
524288	32	44198	38009	42866	42317	3870	26257	719717
524288	64	41615	38454	43949	44039	6741	23026	737881
524288	128	41948	39532	33744	34066	10292	24893	760773
524288	256	44669	38385	33796	33996	17062	30767	757784
524288	512	44651	38109	25343	33992	23640	35271	756082
524288	1024	41366	37215	33839	33696	28581	36505	741026
524288	2048	41689	38172	33480	33904	31290	38624	731653
524288	4096	41902	38527	33791	34054	32871	37806	647198
524288	8192	43780	39340	33760	34014	38135	38636	539037
524288	16384	41839	36566	33855	33992	41742	38052	438904
1048576	32	48647	43457	51974	50729	3741	13580	736269
1048576	64	47598	44880	52028	49729	6651	16863	752155
1048576	128	47903	44588	40497	40310	10630	23188	762227
1048576	256	46938	43399	40479	40324	17844	27060	767056
1048576	512	48835	44567	39074	40251	25782	31984	772620
1048576	1024	46947	43869	40073	39190	32597	35400	764993
1048576	2048	48131	44802	40459	40255	36399	36181	773598
1048576	4096	47444	42581	40470	39662	39282	39992	690227
1048576	8192	48339	42919	40092	39778	44592	40389	587334
1048576	16384	46544	41356	39724	39970	47129	41072	528597

Internal Hard Disk eCryptfs (Linux)

KB	reclen	write	rewrite	read	reread	rand read	rand write	record rewrite
64	32	44848	70256	626961	680790	666666	37405	60153
64	64	40074	71987	552153	571850	587155	57399	72155
128	32	56886	99292	740167	775885	761904	97043	182048
128	64	67299	102568	688307	706935	711111	49174	124036
128	128	62715	94956	592313	621118	624390	91168	102241
256	32	57606	114240	789976	825958	810126	112676	305871
256	64	71348	69734	750554	802211	792569	115471	211214
256	128	88426	124390	703160	735445	739884	125613	176067
256	256	80048	98616	595425	628921	635235	91330	106581
512	32	97767	138264	821985	853490	839344	141710	392672
512	64	100293	135274	819128	846253	837970	138079	338825
512	128	104190	133091	787622	817834	811410	116788	278119
512	256	106469	140545	718087	739973	752941	134489	203101
512	512	102915	127138	596090	629718	630541	131181	138606
1024	32	109028	95016	840691	865576	855471	101155	580849
1024	64	94256	96640	849763	875995	863406	107371	365308
1024	128	108867	129176	825064	857584	856187	130779	402357
1024	256	105067	84845	791961	820486	818545	147699	300376
1024	512	108923	90628	715594	738799	741491	93124	241050
1024	1024	106877	91895	588497	603380	592249	127015	126045
2048	32	68515	26049	667553	868560	854757	67877	541529
2048	64	16544	58821	852989	880132	878592	122546	530703
2048	128	100200	44632	859023	875256	877463	148707	430618
2048	256	111310	34975	842097	850139	856903	88466	336072
2048	512	93150	70358	792236	795668	802193	105068	244481
2048	1024	73331	70547	691866	710107	705234	15853	223678
4096	32	66578	65290	861064	871316	857441	74892	624115
4096	64	63653	64777	878947	885818	877275	30390	528724
4096	128	65367	65748	880299	886399	888310	71231	615760
4096	256	64267	64947	867238	885429	878216	64980	481490
4096	512	58389	65269	844890	858148	843840	64696	376328
4096	1024	40823	23601	684143	781666	778855	68317	275121
8192	32	50931	59066	867991	878132	856724	54921	687706
8192	64	54356	56648	782199	895398	886100	57655	678650
8192	128	51936	59443	895401	901199	891015	55032	617706
8192	256	35253	56341	890054	902307	899527	57267	624769
8192	512	51261	56109	770064	884656	886676	58053	509703
8192	1024	50673	59932	826050	832523	830158	55766	428698
16384	32	50688	54225	802229	887592	861816	53872	679580
16384	64	41947	55531	891840	902749	886100	53752	687216
16384	128	48575	52453	898144	910882	906144	53144	735763
16384	256	47684	55482	824314	910478	907097	53476	718815
16384	512	41091	57108	811692	902804	902699	49789	684689
16384	1024	51297	39163	835155	871633	865321	52888	619315
32768	32	44129	51384	866282	890361	855204	52674	727290
32768	64	47809	49060	826829	905343	890943	53107	744746
32768	128	42704	52801	893422	915179	906445	53050	707701
32768	256	44062	51498	897679	916819	916074	52916	753390
32768	512	45041	50756	891498	915129	910272	53048	724348
32768	1024	45215	52303	816891	891550	889130	53451	685522
65536	32	48386	52263	878134	891560	861261	51949	734214
65536	64	48947	52269	892277	905344	889057	52100	753660

65536	128	48263	52285	899329	918358	908518	52314	764955
65536	256	49091	52237	905344	921509	910842	52308	763396
65536	512	48451	51920	879630	920101	917806	52513	753494
65536	1024	48718	52278	888237	900305	899503	52464	732392
131072	32	48269	51675	882651	893092	862622	51662	740653
131072	64	48556	51582	896316	903272	893384	51729	694355
131072	128	48663	51537	908417	917467	912027	51829	768949
131072	256	48654	51660	910431	919590	918662	51793	773409
131072	512	48641	51659	907845	924468	920960	51980	770572
131072	1024	48521	51879	895772	907380	907851	52046	757974
262144	32	50156	52744	881752	892900	861901	49628	741783
262144	64	50386	52656	898048	908011	887892	50095	762072
262144	128	50459	52653	865858	918704	910057	50506	771157
262144	256	50415	52654	870433	922590	919052	50930	776545
262144	512	50484	52748	881328	919690	920569	52087	779010
262144	1024	50432	52662	895930	914141	911916	52604	720744
524288	32	52059	53926	883783	892513	859249	13439	739945
524288	64	52680	53970	881661	903016	892049	18442	743299
524288	128	51208	53701	909806	918199	907256	23917	762657
524288	256	52628	53971	914311	920907	917776	29514	777033
524288	512	52713	54015	913325	898541	921737	36877	775186
524288	1024	52674	53972	903301	915499	910317	43388	779188
1048576	32	53645	52801	1254924	1267084	1208656	8945	725844
1048576	64	54053	52755	1282600	1287729	1258225	12622	750353
1048576	128	54055	52597	1297347	1312240	1293276	18068	758687
1048576	256	53722	52982	1307060	1256503	1309731	24073	778798
1048576	512	54093	52983	1307332	1323007	1313723	31304	782538
1048576	1024	54024	52979	1282394	1298139	1282111	37813	771806

Bibliography

[1] Matthew Jones. Cybercrime becoming more organised. 18 Sep 2006. URL http://www.ioltechnology.co.za/article_page.php?iSectionId=2885&iArticleId=3442667.

[2] FRANKFURT. Axel springer hit by new german data leak scandal. 18 Oct 2008. URL http://www.reuters.com/article/idUSTRE49H1GH20081018.

[3] Danny Shaw. T-mobile staff sold personal data. 17 Nov 2009. URL http://news.bbc.co.uk/2/hi/8364421.stm.

[4] Juliet Eilperin. Hackers steal electronic data from top climate research center. 20 Nov 2009. URL http://www.washingtonpost.com/wp-dyn/content/article/2009/11/20/AR2009112004093.html.

[5] Web application security statistics. 2008. URL http://projects.webappsec.org/Web-Application-Security-Statistics.

[6] Dorothy E. Denning and Peter J. Denning. Data security. *ACM Comput. Surv.*, 11(3):227–249, 1979. ISSN 0360-0300. URL http://doi.acm.org/10.1145/356778.356782.

[7] Chris Savarese and Brian Hart. The caesar cipher. 2002. URL http://www.cs.trincoll.edu/~crypto/historical/caesar.html.

[8] Fred Cohen. A short history of cryptography. Copyright(c), 1990, 1995. URL http://all.net/BOOKS/IP/cHAP2-1.HTML.

[9] Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical Unix & Internet Security, 3rd Edition*. O'Reilly Media, Inc., 2003. ISBN 0596003234.

[10] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982. ISBN 0-201-10150-5.

[11] SA Vanstone AJ Menezes, PC Van Oorschot. *Handbook of applied cryptography*. CRC Press, 1996. ISBN 0849385237, 9780849385230.

[12] Encryption algorithms. URL http://www.mycrypto.net/encryption/crypto_algorithms.html.

[13] RSA Laboratiories. Pkcs#1: Rsa cryptography standard. 2002. URL ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

[14] Data encryption standard (des). 1999. URL http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[15] Dongyue Xue Jia Chen and Xuejia Lai. An analysis of international data encryption algorithm (idea) security against differential cryptanalysis. *Wuhan University Journals Press, China*, 2008. URL http://www.springerlink.com/content/l178210gl737488u/.

[16] K.H.Tsoi M.P.Leong, O.Y.H.Cheung and P.H.W.Leong. A bit-serial implementation of the international data encryption algorithm idea. 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.5852&rep=rep1&type=pdf.

[17] *Description of a new variable-length key, 64-bit block cipher (Blowfish)*, volume Volume 809/1994, pages 191–204. Springer Berlin, Heidelberg, 1994. doi: 10.1007/3-540-58108-1_24.

[18] Dr. Dobb's Journal B. Schneier. The blowfish encryption algorithm. 1995. URL http://www.schneier.com/paper-blowfish-oneyear.html.

[19] D. Whiting D. Wagner C. Hall N. Ferguson B. Schneier, J. Kelsey. Twofish: A 128-bit block cipher. 1998. URL http://www.schneier.com/paper-twofish-paper.pdf.

[20] CARLISLE M. ADAMS. Constructing symmetric ciphers using the cast design procedure. Ottawa, Canada, 1998. URL http://jya.com/cast.html.

[21] J. Daemen and Rijmen. *The Design of Rijndael: AES-the advanced encryption standard*. Springer-Verlag New York, Inc., 2002.

[22] A Publication of the BITS Security Working Group. Enterprise key management. May 2008. URL http://www.bitsinfo.org/downloads/Publications%20Page/BITSEnterpriseKeyManagementMay2008.pdf.

[23] Arshad Noor. Securing the core with an enterprise key management infrastructure (ekmi). In *IDtrust '08: Proceedings of the 7th symposium on Identity and trust on the Internet*, pages 98–111, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-066-1. URL http://doi.acm.org/10.1145/1373290.1373303.

[24] Trusted Computing Group. Enterprise key management. 2006. URL http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[25] Frank Wells Sudia. Enhanced cryptographic system and method with key escrow feature. 1998. URL http://www.google.no/patents?id=cB8pAAAAEBAJ&printsec=description&zoom=4#v=onepage&q=&f=false.

[26] Lorrie Cranor. Introduction to Cryptography and the Clipper Chip Controversy. 1995. URL http://lorrie.cranor.org/pubs/crypt1.html.

[27] Red Hat Documemtation. Overview of LUKS. 2008. URL http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/html/Installation_Guide/ch29s02.html#id4702552.

[28] Clemens Fruhwirth. New methods in hard disk encryption. 2005. URL http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf.

[29] Clemens Fruhwirth. Tks1: An antiforensic two level and iterated key setup scheme. 2004. URL http://clemens.endorphin.org/TKS1-draft.pdf.

[30] David Braun. Disk encryption howto. 2004. URL http://Linuxreviews.org/howtos/security/Disk-Encryption-HOWTO/en/index.html.

[31] Bill Bosen. File level vs. full drive encryption. 2008. URL http://www.itsecurityjournal.com/index.php/Latest/File-Level-vs.-Full-Drive-Encryption.html.

[32] Christian Stble Marcel Winandy. Ahmad-Reza Sadeghi, Michael Scheibel. Design and implementation of a secure Linux device encryption architecture. 2006. URL http://www.trust.rub.de/media/trust/publications/SaScStWi2006.pdf.

[33] Michael Austin Halcrow. ecryptfs: An enterprise-class cryptographic filesystem for Linux. In *in Proceedings of the Linux Symposium*, page 201218, Ottawa, Canada, July 2005. URL http://www.dubeyko.com/development/FileSystems/eCryptfs/ecryptfs.pdf.

[34] Mike Halcrow. ecryptfs: a stacked cryptographic filesystem, April 2007. URL http://www.Linuxjournal.com/article/9400.

[35] Christophe Saout. dm-crypt: a device-mapper crypto target. 2004. URL http://www.saout.de/misc/dm-crypt/.

[36] TrueCrypt Foundation. TrueCrypt user's guide. 2005. Retrieved 2007. URL http://security.ngoinabox.org/Programs/Security/Encryption%20Tools/TrueCrypt/TrueCrypt%20User%20Guide.pdf.

[37] TrueCrypt: Free Open-source on-the-fly Encryption. URL http://www.truecrypt.org/docs/.

[38] Byron Hynes. Advances in bitlocker drive encryption. 2008. URL http://technet.microsoft.com/en-us/magazine/cc510321.aspx.

[39] Martin Kiaer. A best practice of how to configure bitlocker. Jan 2007. URL http://www.windowsecurity.com/articles/Best-practice-guide-how-configure-BitLocker-Part2.html.

[40] Microsoft TechNet Documentation. Chapter 2: BitLocker Drive Encryption. April 04, 2007. URL http://technet.microsoft.com/en-gb/library/cc162804.aspx.

[41] Randy Muller. How it works: Encrypting file system. 2006. URL http://technet.microsoft.com/en-us/magazine/2006.05.howitworks.aspx.

[42] Russell Coker. Bonnie++ Website. URL http://www.coker.com.au/bonnie++/.

[43] IOzone Filesystem Benchmark. Last Updated: 28th Oct, 2006. URL http://www.iozone.org/.

[44] Chris Mason. Seekwatcher I/O. URL http://oss.oracle.com/~mason/seekwatcher/.

[45] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, Montgomery, J. A., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Norm, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, . Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox. Gaussian 09 Revision A.02, Gaussian Inc. Wallingford CT 2009.

[46] Microsoft Corporation, 2003. Windows Server 2003 Resource Kit Tools.

[47] R. J. Bartlett, G. D. Purvis III, Many-body perturbation-theory, coupled-pair many-electron theory , importance of quadruple excitations for correlation problem, Int. J. Quantum Chem.,14 (1978) 561-81.

[48] G. E. Scuseria, C. L. Janssen,, H. F. Schaefer III. An efficient reformulation of the closed-shell coupled cluster single, double excitation (CCSD) equations, J. Chem. Phys., 89 (1988) 7382-87

[49] Matplotlib Documentation. March, 2010. URL http://matplotlib.sourceforge.net/contents.html.

[50] Enthought Python Distribution. URL http://www.enthought.com/products/getepd.php.