NTNU

Norwegian University of
Science and Technology

# Disk Encryption

Scrutinizing IEEE Standard 1619\XTS-AES

**Adnan Vaseem Alam**

Master of Science in Communication Technology

Submission date: June 2009

Supervisor: Danilo Gligoroski, ITEM

Norwegian University of Science and Technology
Department of Telematics

# Problem Description

'IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices' (IEEE Standard 1619-2007) is a new IEEE standard relating to narrow-block disk encryption. The standard was approved in December 2007 by the IEEE-SA Standards Board and describes the XTS-AES encryption algorithm and a standard for an XML-based key-export format.

In this thesis, the student will study principles and investigate methods used in disk encryption, elaborate on IEEE Standard 1619-2007, provide a security assessment of XTS-AES, and lastly perform a performance benchmark on hard drives using full disk XTS-AES encryption.

Assignment given: 15. January 2009
Supervisor: Danilo Gligoroski, ITEM

# Abstract

Disk encryption has changed during the last decade from being a mechanism only used by corporate executives and government agencies for their top secret information, to become trivially feasible for everyone to utilize. One of the forces that have been driving this development is the steady flow of new cryptographic primitives such as tweakable narrow- and wide-block ciphers specifically designed for disk encryption implementations. One such tweakable narrow-block cipher is XTS-AES, which is standardized in IEEE Std 1619 and very recently accepted by NIST as an approved mode of operation for AES under FIPS-140.

In the first part of this thesis, we study principles and investigate methods used in disk encryption. We present the different implementation types of disk encryption (hardware-based versus software-based, wide-block versus narrow-block, authenticated versus transparent), commonly discussed modes of operations (LRW, XEX, MCB, CMC, EME, XCB), and briefly review some open-source software implementations of disk encryption (TrueCrypt, FreeOTFE, dm-crypt).

In the second part of this thesis we provide a thorough examination of XTS-AES, describing both its security and real-world performance. To our knowledge, this is the first scientific work to provide an elaborate description of XTS-AES while also assessing its real-world performance. Our work show that introducing XTS-AES-256 full system disk encryption using TrueCrypt 6.1a on Windows yield a decrease in write and read speed of up to $-35$ % (average for Windows XP, Windows Vista, and Windows 7 Beta). Further, our results also show that disk operations that uses approximately 2 % of the CPU resources when no disk encryption is present, takes up to 50 % of the CPU resources when full system disk encryption is deployed.

# Preface

This master's thesis is the final outcome of the author's master studies at The Norwegian University of Science and Technology (NTNU). The assignment was given by Professor Danilo Gligoroski at the Department of Telematics, NTNU. The research and writing were performed over a five-month period (February-June 2009) in Trondheim, Norway.

This thesis is the result of an extensive research and the corroboration of many sources. Finding concrete and reliable information about disk encryption has proven to be a struggle, but I have hopefully created a comprehensive guide to the topic of disk encryption and specifically the XTS-AES cipher mode. Conducting research on and experimentation with disk encryption has been an enriching experience and I am very pleased to have had the opportunity to scrutinize the state of art narrow-block cipher mode XTS-AES.

**Last minute update, May 26 2009:** NIST has accepted XTS-AES as an approved mode of operation for AES [27]. NIST plans to produce a draft Special Publication that uses IEEE Std 1619-2007 as the reference for XTS-AES, along with some general guidance. However, no official press release from NIST has yet been published.

## Acknowledgements


- My tutor, Professor Danilo Gligoroski for his encouraging feedback and kind advice.
- NTNU staff for their generous help in providing me with necessary equipment, software and licenses.
- My colleagues, family, and friends for their encouragement throughout my master studies.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| **AES** | Advanced Encryption Standard |
| **Base64** | Content encoding as specified in RFC 3548 |
| **CBC** | Cipher Block Chaining |
| **CCM** | Counter with CBC-MAC |
| **CFB** | Cipher FeedBack |
| **CMAC** | Cipher-based Message Authentication Code |
| **CMC** | CBC-Mask-CBC |
| **CPU** | Central Processing Unit |
| **CTR** | Counter |
| **CTS** | Ciphertext Stealing |
| **DMTF** | Distributed Management Task Force |
| **DTD** | Document Type Definition |
| **ECB** | Electronic CodeBook |
| **EME** | Encrypt-Mask-Encrypt |
| **ESSIV** | Encrypted Salt-Sector Initialization Vector |
| **FIPS** | Federal Information Processing Standard |
| **GCM** | Galois/Counter |
| **GF** | Galois Field |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IETF** | Internet Engineering Task Force |
| **INCITS** | InterNational Committee for Information Technology Standards |
| **IP** | Intellectual Property |
| **ISO/IEC JTC1** | International Organization for Standardization/International Electrotechnical Commission Joint Technical Committee |

| | |
|---|---|
| **IV** | Initialization Vector |
| **KB** | Kilobyte |
| **LRW** | Liskov, Rivest, Wagner |
| **MAC** | Message Authentication Code |
| **MBR** | Master Boot Record |
| **MCB** | Masked CodeBook |
| **NIST** | National Institute of Standards and Technology |
| **OCB** | Offset CodeBook |
| **OFB** | Output FeedBack |
| **OS** | Operating System |
| **P1619.1** | Authenticated Encryption project of SISWG |
| **P1619.2** | Wide-block Encryption project of SISWG |
| **P1619.3** | Key Management Infrastructure project of SISWG |
| **P1619** | Narrow-block Encryption project of SISWG |
| **PDA** | Personal Digital Assistant |
| **PMAC** | Parallelizable MAC |
| **PRP** | Pseudo Random Permutation |
| **SISWG** | Security in Storage Working Group |
| **SNIA** | Storage Networking Industry Association |
| **Std** | Standard |
| **TCB** | Tweakable CodeBook |
| **TCG** | Trusted Computing Group |
| **XCB** | Extended CodeBook |
| **XEX** | XOR-Encrypt-XOR |
| **XML** | Extensible Markup Languange |
| **XOR** | Exclusive OR |
| **XTS** | XEX-based Tweakable CodeBook mode with Ciphertext Stealing |

# Chapter 1

# Introduction

*There are two types of encryption: one that will prevent your sister from reading your diary and one that will prevent your government.*

**Bruce Schneier**

Computer data or digital information is one of the most sensitive and important assets of people, businesses and organizations today. This is a result of the prolonged process of everything and everyone going digital, which started many years ago. Although the digital information age has brought on minimized paperwork, increased efficiency, automation, and increased overall productivity, one often neglected byproduct is also the need for increased security awareness.

## 1.1 Motivation

As the dependency on computers has increased, so has the number of portable devices like laptops and PDAs. With these, the chances of data theft, system compromise and intrusion activities have also increased greatly. In most cases, the actual data is significantly more valuable than the asset it is stored on, and unwanted disclosure of that data can thus be very damaging.

One solution is to use disk encryption to encrypt all the data on the disk, effectively reducing the risk of unwanted disclosure. Disk encryption has changed during the last decade from being a mechanism only used by corporate executives and government agencies for their top secret information, to become trivially feasible for everyone to utilize. One of the forces that have been driving this development is the steady flow of new cryptographic primitives such as tweakable narrow- and wide-block ciphers specifically designed for disk encryption implementations. One such tweakable narrow-block cipher is XTS-AES, which in December 2007 was standardized as a part of IEEE's newest standard on the area of narrow-block encryption, *IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices* (IEEE Std 1619-2007).

## 1.2 Objectives

The objectives of this thesis can be summarized as follows:

- Present a detailed introduction to the topic of disk encryption
- Give a careful examination of IEEE Std 1619-2007
- Provide a security assessment of XTS-AES
- Perform real-world performance benchmarks of XTS-AES

## 1.3 Research Methodology

Our research can be divided into four phases, based on the objectives stated in the previous section. *First*, we will perform a comprehensive study of the general topic of disk encryption. This will provide us with the required knowledge needed for the second objective of this thesis, namely the task of elaborating on IEEE Std 1619. *Next*, we will assess the security of XTS-AES and find attacks that may be applicable to XTS-AES. *Then*, we will experiment with some open-source implementations of the XTS-AES cipher, finding the most suitable implementation to use in the *last* phase of our research, namely the performance benchmark showing how disk encryption using XTS-AES affects the hard disk performance of a computer. The test methodology used during the performance benchmark is presented in section 5.5.

In the first part of our research, our main sources of information will include books, papers, standardization documents, and accredited web pages on the topic of cryptography. For the second part we will be using the standardization document IEEE Std 1619 [CC07]. But, since this is a relatively new standard, we will also use other sources of information to understand and detect misprints and unfavorable explanations. Thus, we will use the P1619 Task Group Email Archive [19] and comments from the cryptographic community [26] for errata and supplementary information about the XTS-AES cipher. The sources used in the security analysis of XTS-AES will also be the IEEE Std 1619 and P1619 Email Archive, along with other relevant papers where the security of XEX and XTS-AES is discussed [Rog04, LM08, Min07].

## 1.4 Outline

The remainder of this thesis consists of six main chapters in addition to references, web resources, and appendices. All the test results included in this document are available electronically with the thesis, along with the software used in the performance benchmark.

- Chapter 2 gives a comprehensive introduction to topic of disk encryption, providing the reader with the necessary background to recognize different types of disk encryption and know the most common modes of operation for disk encryption.

- Chapter 3 provides a careful examination of IEEE Std 1619, describing both the XTS-AES cipher mode and XML-based key export format.
- Chapter 4 presents the reader with a security assessment of XTS, comprising a review of the computational security of XTS-AES, a synopsis of comments submitted by the cryptographic community regarding XTS-AES, and a discussion on attacks applicable to XTS-AES.
- Chapter 5 specifies the procedure, test bench, test cases, and testing methodology used in the performance benchmark.
- Chapter 6 presents and discusses the results obtained from performance benchmark described in the previous chapter.
- Chapter 7 summarizes this thesis and its findings.
- Appendix A gives step-by-step instructions on how full system disk encryption was employed during the performance benchmark.
- Appendix B lists the exhaustive result sets with relevant statistics.
- Appendix C lists the contents of the attached ZIP file.

Throughout this thesis, we have distinguished between printed references and web resources to clearly state the origin of our sources of information. Printed references are cited using alphanumerical values and web resources are cited using numerical values. Web resources that did not provide an explicit date of creation has been given the date of our visit. Additionally, we emphasize that the usage of first person plural (i.e. we), is only due to common convention and should therefore be interpreted as the author.

# Chapter 2

# Background on Disk Encryption

*The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable.*

**The Art of War, Sun Tzu**

This chapter will give a comprehensive introduction to relevant theory to understand how disk encryption works. First, we will start with an introduction to basic cryptographic elements used throughout this thesis. Next, we will address disk encryption specifically, describing its various implementation types. Then, we will review some modes of operation especially designed for disk encryption. Lastly, we will describe the software-based disk encryption applications that at the time of writing support the XTS-AES cipher mode.

## 2.1 Basic Cryptography

Cryptography, derived from the Greek *kryptó* "hidden" and *gráfo* "to write", is the ancient science and art of hiding the content of a message from prying eyes. Although now considered a branch of modern number theory and computer science, it was originally literarily done by hand as early as 4000 years ago [Kah97]. The Egyptians used substitution ciphers to substitute hieroglyphs with less common varieties of hieroglyphs in inscriptions on grave chambers, presumably in order to obfuscate the meaning of the inscriptions for people who did not know how to reverse the substitutions.

Modern cryptography is more complex, and the following subsections aim towards defining basic cryptographic elements used throughout this thesis.

### 2.1.1 Terminology

Cryptography uses its own terminology that we will attempt to follow throughout this thesis. Although most of the terminology might be intuitive, we want to prevent any erroneous conceptions by providing a description of the terminology used in this thesis.

First of all, *encryption* is the process of encoding a message to hide its content, while *decryption* is the corresponding reverse operation. The mathematical notations for these two operations are expressed in Equation 2.1 and 2.2 respectively, where $p$ denotes the intelligible plaintext and $c$ denotes the obfuscated ciphertext.

$$E(p) = c \tag{2.1}$$

$$D(c) = p \tag{2.2}$$

By replacing $c$ with $E(p)$ in Equation 2.2, we can easily see that decryption of encrypted content work as follows:

$$D(E(p)) = p \tag{2.3}$$

As ciphers usually use *keys* to ensure their secrecy, we indicate encryption and decryption with the key $k$ as:

$$E_k(p) = c \tag{2.4}$$

$$D_k(c) = p \tag{2.5}$$

Furthermore, *symmetric key cipher* will denote a cryptographic algorithm using the same key for both encryption and decryption, and *plaintext* and *ciphertext* the corresponding pair of plain and enciphered message. In future equations and figures these two objects are denoted as $P_i$ for plaintext and $C_i$ for ciphertext, where $i$ is the index. Moreover, *cipher mode* will often be used as short for *block cipher mode of operation*. Lastly, a *cryptosystem* will refer to a system consisting of a symmetric key cipher, all possible plaintext-ciphertext pairs, and the corresponding key.

#### Main Cryptographic Goals

Menezes et al. [MvOV01] defines four mechanisms or services cryptography attempts to provide:

1. Confidentiality is a service *"used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible."*

2. Data integrity is a service *"which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution."*

3. Authentication is a service *"related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent etc."*

4. Non-repudiation is a service *"which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. [...] A trusted third party is needed to resolve the dispute."*

It is surely possible to identify other services provided by cryptography as well, but these would probably be based upon one or more of the goals above. For example, although disk encryption is a cryptographic service it self, it is employed through the use of one or more of the mechanisms above. Hence, the borders between cryptographic services are not absolute, and overlaps occur.

### Cryptographic Attack Models

Typically, the objective of an attack on an encryption system is to recover the key in use, rather than simply recover the plaintext of one single ciphertext. This is consistent with Kerckhoffs' assumption [Ker83], which states that the strength and security of an encrypted message should lie in keeping the key secret, not in assuming that the algorithm used to encrypt it is unknown to the adversary. Thus, we assume that the adversary's main goal is to deduce the key.

We consider the following two approaches for attacking a conventional encryption scheme in order to deduce the key [Sta06]:

- A *cryptanalytic attack* rely on the nature of the algorithm in addition to partial or complete knowledge to the general characteristics of the plaintext or even some plaintext-ciphertext pairs. This type of attack exploits the construction of the algorithm to deduce the key being used.

- A *brute-force attack* is a more primitive attack type. This attack is carried out by trying every possible key on a piece of ciphertext until an intelligible translation can be made. On average, half of all possible keys must be tried to achieve success.

If either of these types of attack succeeds in deducing the key, the effect is potentially grave as all future and past plaintexts encrypted with that key are compromised. Cryptanalytic attacks can further be divided into various attack models, based on the amount of information known to the adversary [MvOV01]:

- *Ciphertext only attacks* are mounted by trying to recover the key or plaintext from the ciphertext only.

- *Known plaintext attacks* on the other hand are performed if the cryptanalyst has access to one or more plaintext-ciphertext pairs formed with the secret key.

- *Chosen plaintext attacks* are based on the attacker being able to choose the plaintext and retrieve the corresponding ciphertext generated by encryption.
- *Chosen ciphertext attacks* are the opposite, here the cryptanalyst may choose the ciphertext, and obtain the plaintext by decryption.

The difference between these attack models is relevant when we later discuss attacks that may be applicable to XTS-AES in section 4.3.

### Computational Security

When considering the security of an encryption system, one must distinct between issues affecting the theoretical security from those of computational security. The reason for this is that in practice, no adversary can have unlimited computational power [JBSK01]. The security of a practical encryption system need therefore not necessarily depend on the theoretical impossibility of breaking a cipher, but rather on the practical difficulty of launching a successful attack. Throughout this thesis we will use the word "complexity" to describe such difficulty. The complexity of an attack is generally understood to mean the average number of operations used in the attack. For a cryptosystem to be computationally secure this means that the complexity of any successful attack exceeds the computational capability of the adversary. Hence, we need to pay attention to the computational effort needed for a successful attack. To describe computational complexity, we define the following terms:

**Time complexity.** The time complexity denotes the expected time to solve a problem. Note however that expected time in this case is not measured in seconds or years, but rather in problem size. Given a cipher where brute-force key search is the best option, the time complexity is directly dependent on the key size. For example, the time complexity for guessing a 128-bit key is around $2^{128-1} = 2^{127}$. The actual time it will really take to guess it depends on the adversary's resources, methodology, and luck.

**Space complexity.** Space complexity on the other hand, refers to the space requirements (i.e. data input) needed to solve a problem. Just like time complexity, space complexity is also measured in problem size. Consider the following two examples to distinguish time and space complexity. In an exhaustive key-search attack, the amount of input data needed for the attack is an arbitrary number of ciphertext blocks, which is generally a very small number in comparison with the number of operation needed to test every possible key. Therefore, the complexity of such an attack is clearly time complexity. An opposite example is differential cryptanalysis[1], where the amount of input data needed (i.e. ciphertext-plaintext pairs) dominates, while the number of computations used in the attack is relatively small. Hence, the complexity of such an attack is space complexity.

[1] Differential cryptanalysis is a general form of cryptanalysis relating to the study of how differences in an input can affect the resultant difference at the output [MvOV01]. In the case of a block cipher, it refers to a set of techniques for tracing differences through the network of permutations and substitutions, discovering where the cipher exhibits non-random behavior, and exploiting such properties to recover the secret key [vT06].

### 2.1.2 Cryptographic Ciphers

The following paragraphs will give a superficial introduction to the three main types of ciphers in today's cryptography; block ciphers, stream ciphers and the fairly new tweakable block ciphers.

#### Block Ciphers

In cryptography, a block cipher is a symmetric key cipher that operates on a fixed-length group of bits (hence, the term *block*). A conventional block cipher takes a key $K \in \{0,1\}^k$ and a fixed-length message $M \in \{0,1\}^n$ as input, and produces a fixed-length ciphertext $C \in \{0,1\}^n$. The signature of a block cipher is defined in Equation 2.6.

$$E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n \tag{2.6}$$

Most block ciphers have similar structures (the Feistel structure being the most popular one [Sta06]), which normally consist of a number of identical round of processing. In each round, a substitution or permutation is performed to obfuscate the original content. When doing so for a number of rounds, including the use of a secret (i.e. key), a block cipher is able to provide confidentiality by diffusion and confusion[2]. In fact, diffusion and confusion is so successful in capturing the essence of the desired attributes of a block cipher, that they have become the cornerstone of modern block cipher design [Rob95].

The only block cipher of relevance to this thesis is the Advanced Encryption Standard (AES), which is described in section 2.1.3.

#### Stream Ciphers

Block ciphers can be contrasted with stream ciphers; which operate on individual bits one at the time and the transformation varies during the encryption. Equation 2.7 shows that the signature of a stream cipher is composed by a keystream $K \in \{0,1\}^*$ and variable-length message $M \in \{0,1\}^*$ that produces a variable-length ciphertext $C \in \{0,1\}^*$.

$$E : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \tag{2.7}$$

Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise XOR operation. The generation of the keystream can be independent of the plaintext and ciphertext, yielding what is termed a synchronous stream cipher, or it can depend on the data and its encryption, in which case the stream cipher is said to be self-synchronizing. Most stream cipher designs are for synchronous stream ciphers [22].

[2] The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system [Sha49]. While the mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible, the mechanism of confusion seeks to make the relationship between the statistics of the ciphertext and the value of the key as complex as possible. The main goal of both is to mitigate attempts to discover the key, which is achieved by the use of complex substitution and permutation algorithms [Sta06].

It is worth mentioning that for disk encryption applications, the use of stream ciphers is futile since they require, for their security, the same keystream not be used twice. When also considering the fact that one need to match the amount of keystream with the exact amount of plaintext that is to be ciphered one should clearly identify the need to prohibit this type of ciphers for disk encryption solutions – although we have seen cases of stream ciphers being used for this purpose in the past [Fer06, 6].

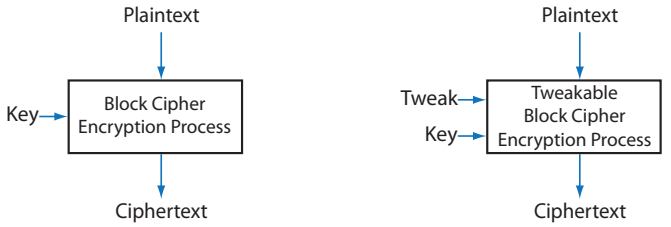
### Tweakable Block Ciphers

Tweakable block ciphers are a new cryptographic primitive proposed by Liskov et al. in their paper "Tweakable Block Ciphers" [LRW02]. While the conventional block cipher process takes two inputs, a tweakable block cipher process takes three inputs; a key $K \in \{0,1\}^k$, a message $M \in \{0,1\}^n$, and a third input called the "tweak" $T \in \{0,1\}^t$. The signature of a tweakable block cipher is defined in Equation 2.8.

$$E : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \rightarrow \{0,1\}^n \tag{2.8}$$

A tweakable block cipher is thus defined quite similar to a conventional block cipher, but makes use of one additional input in the ciphering process. This is depicted in Figure 2.1. Please note that the additional input does not necessarily imply that the internal block cipher construction is changed. Most commonly, block ciphers are used "as is", and the additional tweak value is used in pre- and post-whitening processes. This means that the input and output of the block cipher is combined with the tweak.

Liskov et al. mentions several design goals for the tweakable block cipher design, but the two most important ones are; efficiency, a tweakable block cipher should have the property that changing the tweak should be less costly than changing the key; and security, meaning that even if an adversary has control of the tweak input, the tweakable block cipher must remain secure. The latter implies that the introduction of the tweak is not to provide additional uncertainty to an adversary, only variability [LRW02].




**(a)** *Conventional Block Cipher* **(b)** *Tweakable Block Cipher*

**Figure 2.1:** *Notice that apart from the tweak input, a tweakable block cipher is the same as a conventional block cipher.*

The tweak input in tweakable block ciphers enables a multitude of new modes of operation. [LRW02] describes three such possible modes, and imply that most existing block cipher modes of operation may be re-implemented as tweakable.

### 2.1.3 Advanced Encryption Standard

Advanced Encryption Standard (AES) was published by National Institute of Standards and Technology (NIST) in 2001. AES is the symmetric block cipher that has replaced Data Encryption Standard (DES) as the approved standard for a wide range of applications [34]. AES is an instantiation of the Rjindael block cipher, which was selected as the most suitable design for AES after a 5-year standardization process in which a total of fifteen competing designs were presented and evaluated [30]. AES has been extensively analyzed and is now used worldwide, in fact; AES is the first publicly accessible and open cipher approved by the National Security Agency (NSA) for top secret information [7].

The industry adoption of AES is widespread, and its use in disk encryption solutions is no exception. A quick survey of both hardware-based and software-based disk encryption solutions have namely shown us that AES is the block cipher of choice for the vast majority of the disk encryption solutions available to the public. Although some software-based implementations do provide the possibility to choose between different ciphers, AES is most often the default and recommended choice.

AES is a substitution-permutation network cipher that works on 128-bit data blocks[3], and supports three different key lengths: 128, 192 and 256 bits [DR03]. The ciphering process consists of a certain number of rounds, each consisting of multiple stages. These are stages of permutation and substitution that ultimately provide confidentiality. As depicted in Figure 2.2, the first and last stage of both the encryption and decryption process is always an AddRoundKey stage – the stage mixing the key into the process. The reason for this is that all the other three stages (called SubBytes, ShiftRows and MixColumns) together provide non-linearity, confusion, and diffusion, but would not provide any real security if placed in the beginning or the end; since they do not use the key, and thus are fully reversible.

| **Key size** | 128 bits | 192 bits | 256 bits |
|---|---|---|---|
| **Plaintext block size** | 128 bits | 128 bits | 128 bits |
| **Number of rounds** | 10 | 12 | 14 |
| **Round key size** | 128 bits | 128 bits | 128 bits |
| **Expanded key size** | 1408 bits | 1664 bits | 1920 bits |

**Table 2.1:** *AES parameters [Sta06]*

After the initial AddRoundKey stage, the number of rounds to be performed is based on the length of the key (see Table 2.1). When using AES with a 128-bit key, the master key is expanded into a 1408-bit key by a stage called KeyExpansion, yielding 128 bits for the initial AddRoundKey stage, and additional 1280 bits for 10 rounds of ciphering.

[3] Although Rjindael can be specified with a block size in any multiple of 32 bits between 128-bit and 256-bit, AES is specified to only use a data block size of 128 bits.

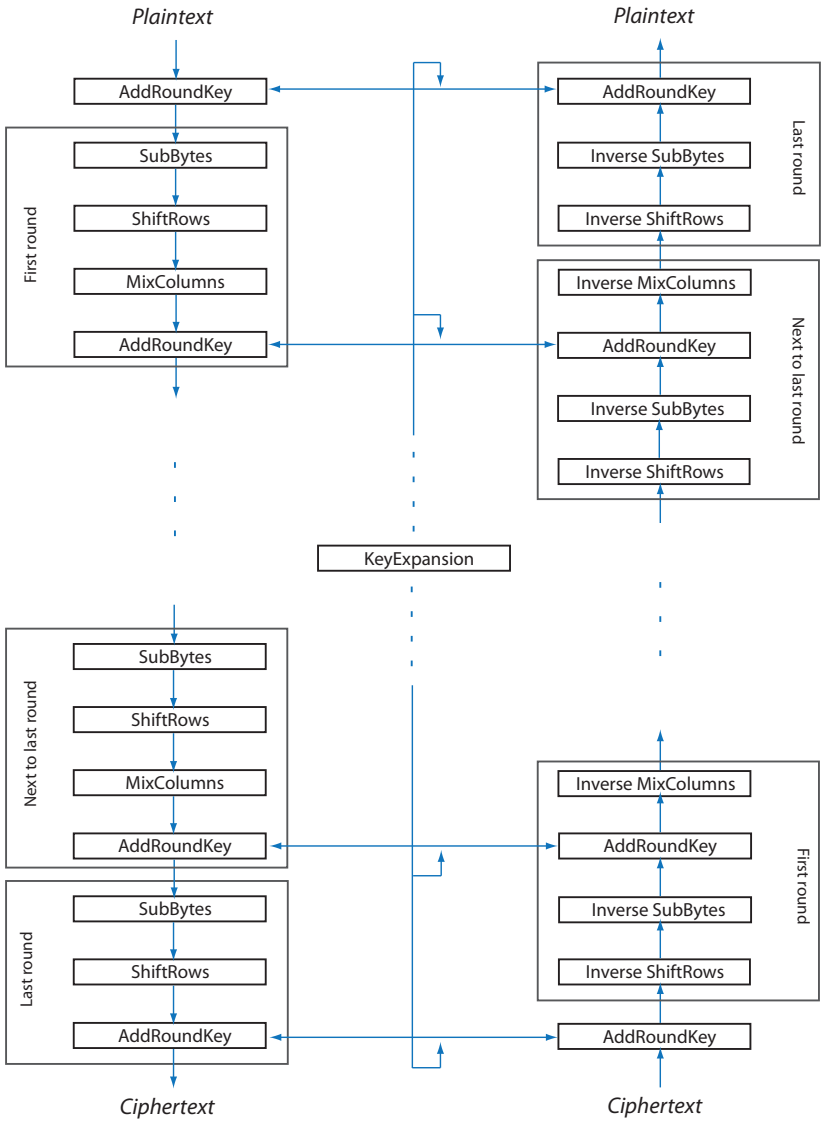



**Figure 2.2:** *AES encryption (left side) and decryption (right side). Figure adapted from [Sta06].*

The following paragraphs will briefly explain each stage of the AES algorithm; KeyExpansion, SubBytes, ShiftRows, MixColumns, and AddRoundKey. Readers familiar with the stages of AES are welcome to skip the next two pages.

In the official AES specification document [oSN], the 128-bit input data block is depicted as a square matrix of bytes. This block is copied into the *State* array, which is modified at each stage. After the final stage, *State* is copied to an output matrix. In the rest of this section, references to the *State* are the array which initially contained the plaintext.

### KeyExpansion

The KeyExpansion stage is the very first stage initiated by the AES cipher, and provides round keys for the AddRoundKey stages. The expanded key (i.e. compilation of round keys) is derived by taking the master key as input and expanding this to a size that is sufficient to provide 128-bit round keys for each of the AddRoundKey stages of the cipher. The KeyExpansion process can be computed ahead of time, as a security-performance tradeoff. Details about how the KeyExpansion stage derives the expanded key can be found in [DR03, oSN].

### SubBytes

In the SubBytes stage, each byte in the *State* is replaced with another according to a lookup table, the Rijndael S-box (found in [DR03]). This operation provides the non-linearity in the cipher. The S-box used, is derived from the multiplicative inverse over Galois Field, GF($2^8$), which is known to have good non-linearity properties [Sta06]. This means that the output cannot be described as a simple mathematical function of the input. The S-box is also chosen to avoid both fixed points (S-box($a$) = $a$) and opposite fixed points (S-box($a$) = $\bar{a}$, where $\bar{a}$ is the bitwise complement of a). The Inverse SubBytes stage is done the same way, but with a different lookup table, the Inverse Rijndael S-box (found in [DR03]).

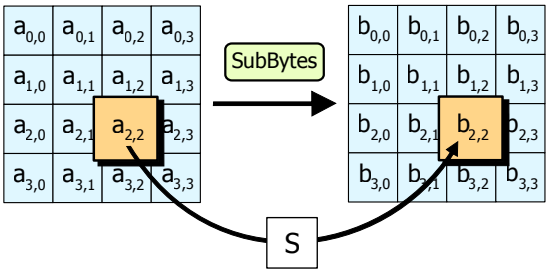



**Figure 2.3:** *AES SubBytes stage*

### ShiftRows

The ShiftRows stage operates, as the name implies, on the rows of the *State*. It cyclically left shifts the bytes in each row by a certain offset, except for the first row (which is left unchanged). On the second row, each byte is shifted one to the left. Similarly, on the third and fourth rows, each byte is shifted by offsets of two and three to the left respectively. This way, each column of the output state of the ShiftRows step is composed of bytes from each column in the input state. The Inverse ShiftRows stage performs the circular shifts in the opposite direction for each of the last three rows.

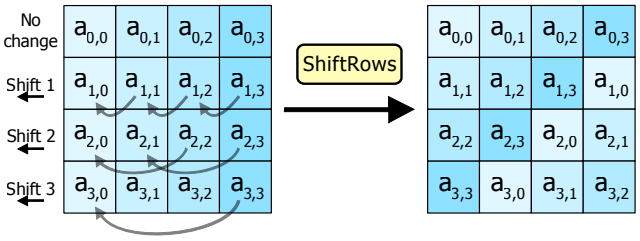



**Figure 2.4:** *AES ShiftRows stage*

### MixColumns

In the MixColumns stage, the four bytes of each column of the *State* are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher. Each column is treated as a polynomial over GF($2^8$) and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $f(x) = 3x^3 + x^2 + x + 2$. The MixColumns and ShiftRows transformations ensure that after a few rounds, all output bits depend on all input bits [35]. The Inverse MixColumns stage reverses the diffusion made by the forward process by treating each column as a polynomial over GF($2^8$) multiplied modulo $x^4 + 1$ with the fixed polynomial $g(x) = 11x^3 + 13x^2 + 9x + 14$. This completely reverses the process since $f(x)$'s inverse is $g(x)$, $g(x) = f(x)^{-1} mod(x^4 + 1)$ [Sta06].

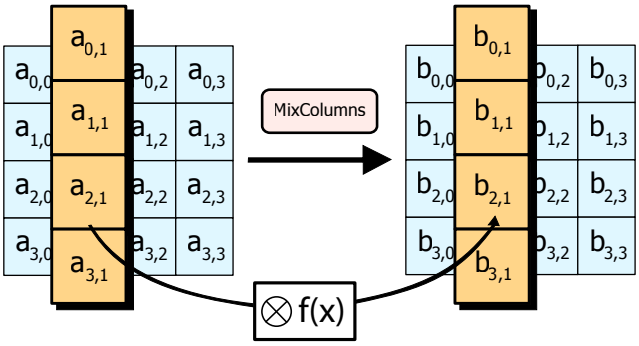



**Figure 2.5:** *AES MixColumns stage*

### AddRoundKey

AddRoundKey is a simple stage where each byte of the *State* is bitwise XORed with a byte of a round key – effectively changing every bit of the *State*. For each time the AddRoundKey stage is performed in either encryption or decryption, a new round key is provided by the KeyExpansion stage. Although the other stages provide essential diffusion and confusion to the cipher, this is the only stage that mixes the actual key material into the *State*. The Inverse AddRoundKey stage is identical, because the XOR operation is its own inverse, $(a \oplus k) \oplus k = a$.

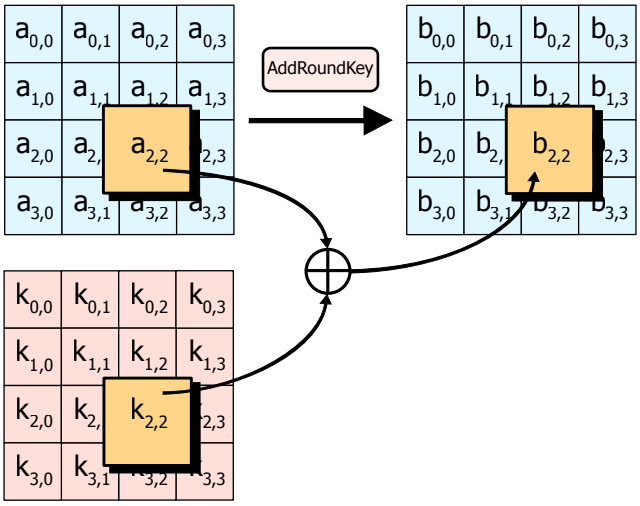



**Figure 2.6:** *AES AddRoundKey stage*

### 2.1.4 Block Cipher Modes of Operation

As described in section 2.1.2, a cryptographic block cipher operates on data blocks of fixed length. But, because encrypting the same block of plaintext under the same key always produces the same ciphertext, several modes of operation have been invented. This is a way to add more variability to the encryption process, enabling block ciphers to provide confidentiality for messages of arbitrary block lengths, and optionally also provide authentication capabilities [MvOV01]. These benefits come at a slightly increased computational complexity, and are the reason why these designs are common.

The block cipher modes of operation can be divided into three main categories: confidentiality modes, authentication modes, and combined modes for confidentiality and authentication. The confidentiality modes are focused only on obscuring the original content, while the authentication modes are focused on assuring that the content has not been tampered with undetected. Thus, a combined mode for confidentiality and authentication is focused on both obscuring the original content and assuring that the content has not been tampered with. Notice that the difference between a dedicated authentication mode and hash function (whose purpose is also to assure that data has not been tampered with undetected) is that while the hash function is designed to be hard to invert[4], the authentication mode bases its security on the knowledge of the secret key and the underlying cipher.

| **Name** | **Confidentiality** | **Authentication** |
|---|---|---|
| Electronic CodeBook (ECB) | Y | |
| Cipher-Block Chaining (CBC) | Y | |
| Cipher FeedBack (CFB) | Y | |
| Output FeedBack (OFB) | Y | |
| Counter (CTR) | Y | |
| Cipher-based Message Authentication Code (CMAC) | | Y |
| Counter with CBC-MAC (CCM) | Y | Y |
| Galois/Counter (GCM) | Y | Y |

**Table 2.2:** *NIST approved block cipher modes of operation*

Currently, NIST has officially approved eight block cipher modes of operations (see Table 2.2). Of these, five are confidentiality modes (ECB, CBC, CFB, OFB and CTR), one is an authentication mode (CMAC), and two are combined modes for confidentiality and authentication (CCM and GCM) [38]. However, NIST continuously receives suggestions for new modes of operation, and maintains a list of all proposed modes at [37].

The following paragraphs will give a detailed description of the two general-purpose confidentiality modes that is often used in disk encryption implementations, namely ECB and CBC [Fru05, 6].

[4]Hash functions are designed to be preimage resistant, $2^{nd}$ preimage resistant, and collision resistant.

### ECB: Electronic CodeBook

A cipher is rarely used as originally published. If this is the case, the cipher is said to be used in electronic codebook mode. But, since ECB denotes the absence of any further design, it is also often said that it does not fully qualify as a block cipher mode of operation [vT06]. The term codebook is used because, for a given key there exist a unique ciphertext for every block of plaintext. Therefore, we can image a gigantic codebook in which there is an entry for every possible plaintext pattern showing its corresponding ciphertext. Figure 2.7 and 2.8 depicts the ECB mode encryption and decryption, respectively.

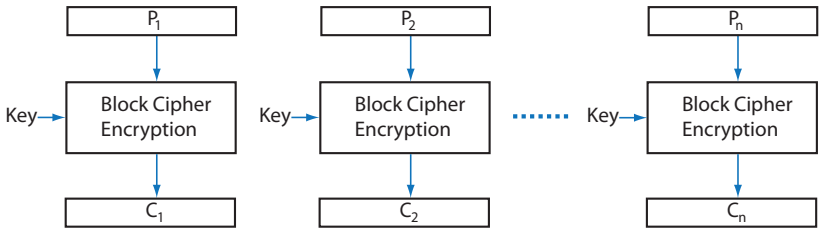



**Figure 2.7:** *ECB mode encryption is expressed as* $C_i = E_k(P_i)$ *for* $1 \leq i \leq n$.

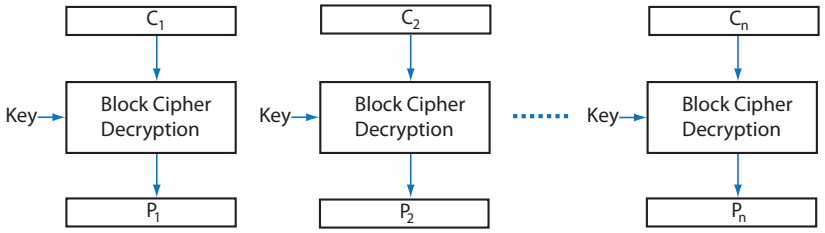



**Figure 2.8:** *ECB mode decryption is expressed as* $P_i = D_k(C_i)$ *for* $1 \leq i \leq n$.

The fundamental disadvantage of this method is the same as the main reason for using modes of operating in the first place, namely that if the same block of plaintext appears more than once in the message, it always produces the same ciphertext. This is unfortunate, because if you know that the message always starts out with certain predefined fields, a cryptanalyst may have a number of known plaintext-ciphertext pairs to work with, and thus be able to recognize repetitive elements. The use of ECB in its original form is therefore often strongly discouraged [MvOV01]. In the past, the ECB mode was sometimes recommended for the encryption of keys; however, authenticated encryption (i.e. using a combined mode for confidentiality and authentication) would be a much better option for this particular application area [vT06].

However, if we slightly modify the ECB mode, or use it as a subroutine – the security concerns related to the ECB mode can be mitigated. Later in this thesis, we will see that there is several cipher modes designed for disk encryption that utilizes ECB – either “as is” or slightly modified.

### CBC: Cipher Block Chaining

Cipher block chaining is a popular block cipher mode that tries to overcome the security deficiencies of ECB. In CBC mode, each block of plaintext is XORed with the preceding ciphertext block before being encrypted. This way, each encrypted block is dependent on all plaintext blocks processed up to that point. To produce the first block of ciphertext, an Initialization Vector (IV) is XORed with the first block of plaintext and then encrypted. This IV must be known to both the sender and receiver, but should at the same time be unpredictable for a third party. By varying this IV, one can ensure that the same plaintext is encrypted into a different ciphertext under the same key, which we have become to know is essential for secure encryption.

Figure 2.9 shows that the input to the encryption function is dependent on the XOR between the plaintext block and the previous ciphertext. Thus, repeating data patterns are not exposed [Sta06].

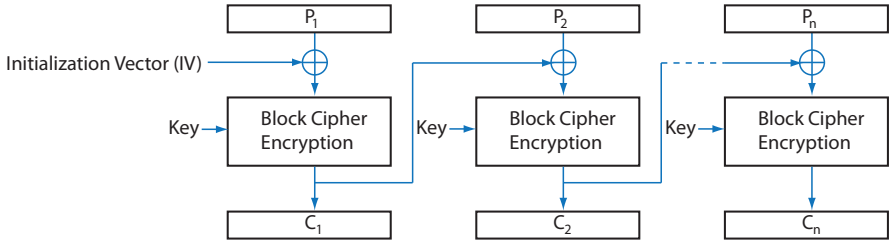



**Figure 2.9:** *CBC mode encryption is expressed as* $C_i = E_k(P_i \oplus C_{i-1})$ *for* $1 \leq i \leq n$, *where* $C_0 = IV$.

Figure 2.10 shows decryption, where the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. Correspondingly, proceeding plaintexts are recovered by XORing the output of the decryption function with the preceding ciphertext.

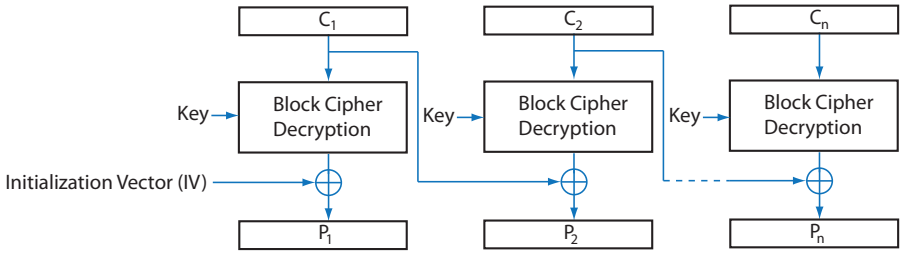



**Figure 2.10:** *CBC mode decryption is expressed as* $P_i = D_k(C_i) \oplus C_{i-1}$ *for* $1 \leq i \leq n$, *where* $C_0 = IV$.

The CBC mode is not completely parallelizable; while the CBC decryption process allows for parallelism and thus random access, the CBC encryption process is strictly a serial operation. But, CBC can be modified to support parallelism for both the encryption process and decryption process. [BDJR97] defines such a variant of the CBC mode that enables parallelism by dividing the plaintext into

$r$ parallel streams and then applying the CBC mode to each of these streams. This however requires the use of $r$ different IV values.

If the CBC mode is to be used for disk encryption, a parallelism-friendly mode like the latter has to be used. By doing so, one avoids that all data blocks are chained together, which in turn would require one to re-encrypt the whole disk for each time new data is written. The way this is avoided is by cutting the CBC chaining for every disk sector and restarting with a new IV, so that it is possible to encrypt sectors individually.

The IV (or IVs) used by the CBC mode can be deduced in many different ways, depending on its application. The following bullet list presents a selection of IV deduction-methods most often associated with disk encryption:

- The most simple and straight-forward option is to choose a *plain-IV*. This implies that the IV is simply a bit representation of $n$ (i.e. number of the last block) padded with zeros to the block size of the cipher used, if necessary. As being the simplest IV mode, it is also the most vulnerable.
- Another way to derive the IV is by using *Encrypted Salt-Sector IV*, short ESSIV[5]. This method derives the IV by combining the disk sector address with the hash of the key (used by the CBC block cipher). As the IV depends on private information (i.e. the key), the IV sequence is not known, and the attacks based on knowledge to the IV cannot be launched.
- The third and highly theoretical way is be to compute the IV by hashing the plaintext from the second block till the last block. This IV deduction-method is called *Plumb-IV*[6]. If a bit changes in one of the plaintext blocks, the first block is influenced by the change of the IV (since the IV is the hash of the other plaintext blocks). As the first encryption affects all subsequent encryption steps due to the chaining process, the whole sector is changed. The obvious weakness of this scheme is its performance, as data has to be processed twice – once for deriving the IV and once for actual ciphering process.

The main drawbacks of the CBC mode is that the encryption process is natively sequential, that messages must be padded to a multiple of the cipher block size prior to encryption, and that its susceptible to a wide range of cryptographic attacks when used with predictable IVs [Fru05]. Additionally, the CBC mode has shown to have a poor avalanche effect[7] [EFD08a]. Despite, CBC is a well-used cipher mode in a wide range of cryptographic applications, but mostly in conjunction with other security enhancing techniques to mitigate its limitations and vulnerabilities.

[5]Clemens Fruhwirth is the author of ESSIV, which was developed for Linux 2.6.10 to counter watermarking attacks [Fru05].

[6]The name of this mode comes from Colin Plumb, who proposed this mode. Bruce Schneier also mentions this construction in [Sch06], but does not give a name for it.

[7]In cryptography, the avalanche effect is the property of a cryptographic algorithm to significantly change an output when the input is changed slightly [Sta06]. For block ciphers or cipher modes this means that a small change in either key, IV or plaintext should cause a drastic change in the ciphertext. The actual term was first used by Horst Feistel, although the concept dates back to at least Shannon's diffusion.

### 2.1.5 CTS: Ciphertext Stealing

Although the block cipher modes just described facilitates encryption and decryption of data lengthier than one block, they are not able to effectively handle data that is not *evenly* dividable into blocks. To address this issue, ciphertext stealing can be used. This is a technique that allows for ciphering of data that is not evenly dividable into blocks without resulting in an expansion of the ciphertext [Sch06].

In principle, any block-oriented confidentiality mode can use CTS, but since stream cipher-like modes can be applied to messages of arbitrary length without padding, they do not benefit from this technique. Thus, the modes of operation that is most often used in combination with CTS, is ECB and CBC [Sch06].

Although there exists multiple ways to implement CTS [vT06], the common for all is that the normal cipher mode procedure is used on all but the last two blocks of data, which are handled differently. Figure 2.11 and the following paragraphs will describe one way to handle the two last blocks of plaintext, denoted $P_{n-1}$ and $P_n$, when CTS is performed on ECB mode. In this description, the following functions are used:

- Head(*data*, $a$) returns the first $a$ bits of the '*data*' string.
- Tail(*data*, $a$) returns the last $a$ bits of the '*data*' string.

**ECB CTS encryption steps (see Figure 2.11a)**

1. The first step is to encrypt $P_{n-1}$ to create $E_{n-1} = E_k(P_{n-1})$. This is equivalent to the behavior of standard ECB mode.

2. The second step is to select the first $M$ bits of $E_{n-1}$ to create $C_n = \text{Head}(E_{n-1}, M)$. The final ciphertext block, $C_n$, is now composed of the leading $M$ bits of the second to last ciphertext block.

3. The third step is to pad $P_n$ with the low order bits from $E_{n-1}$, creating $D_n = P_n \parallel \text{Tail}(E_{n-1}, B - M)$.

4. Finally, the fourth step is to create $C_{n-1}$, which is done by encrypting $D_n$ from the previous step, $C_{n-1} = E_k(D_n)$.

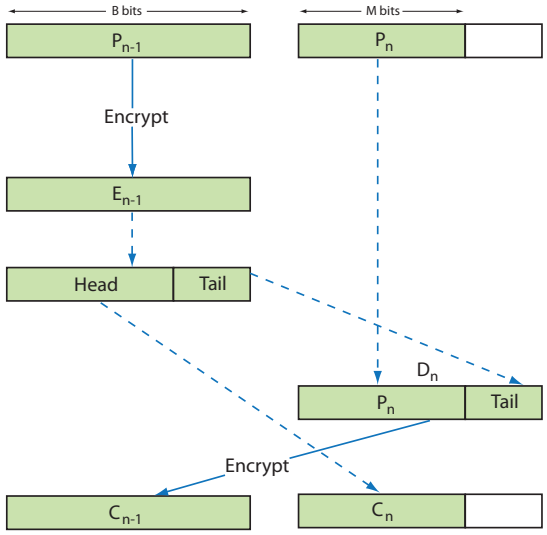
**ECB CTS decryption steps (see Figure 2.11b)**

1. The first decryption step is to decrypt $C_{n-1}$, to find $D_n = D_k(C_{n-1})$. This undoes step 4 of the encryption process.

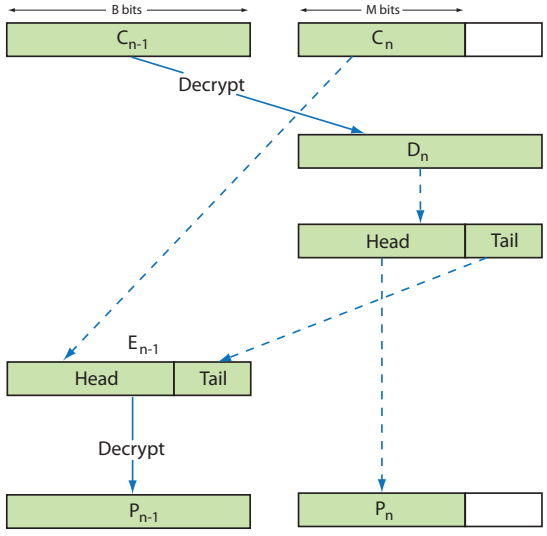
2. The second step is to pad $C_n$ with the extracted ciphertext in the tail end of $D_n$, making $E_{n-1} = C_n \parallel \text{Tail}(D_n, B - M)$.

3. The third step is to select the first $M$ bits of $D_n$ to create the last plaintext block $P_n = \text{Head}(D_n, M)$.

4. The fourth and final step is to decrypt $E_{n-1}$ to create the second to last plaintext block $P_{n-1} = D_k(E_{n-1})$.




**(a)** *Encryption steps*




**(b)** *Decryption steps*

**Figure 2.11:** *The ciphertext stealing technique only affects the last to blocks and in return make the transform length-preserving.*

As we have seen from the latter example, implementing CTS requires buffering of the two most recent blocks of data, so that they can be properly processed at the end of the data stream. Thus, the cost of using CTS is a slightly increased complexity.

## 2.2 Disk Encryption

Encryption of data at rest can be divided into; file-level encryption, folder-level encryption (also called filesystem-level encryption), partition-level encryption, and disk-level encryption, as shown in Figure 2.12. This thesis will not take on the task to elaborate all these different types, but rather focus on disk-level encryption – referred to only as disk encryption from this point on.

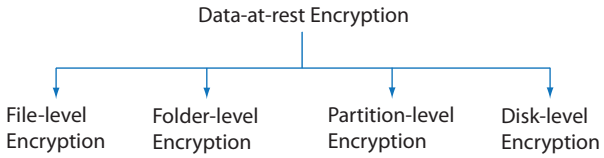



**Figure 2.12:** *Data at rest encryption can be divided into different types, based on granularity.*

We define disk encryption as the cryptographic protection of data at rest when the storage media is a block oriented addressable device (e.g., a hard disk or flash disk). Disk encryption is almost exclusively implemented as an on-the-fly operation; i.e. data is encrypted before being written to the storage device, and decrypted only before use. The following sections will describe the various implementation types of disk encryption.

### 2.2.1 Hardware-based versus Software-based Encryption

The first distinguishable property of disk encryption is whether it is hardware-based or software-based (i.e. layer in which encryption is performed). Although there might be a mistake to even talk about hardware as opposed to software anymore, since almost all of our devices are a mix of software and hardware, it is a certain difference between hardware-based and software-based encryption.

The most evident difference between hardware-based and software-based encryption is that while hardware-based encryption is able to encrypt every single bit on the disk without exception, the software-based encryption techniques are not. The reasons for this are:

- The Master Boot Record (MBR) of a disk contains machine code instructions necessary to mount the disk, and thus has to be left unencrypted. If not, the computer software trying to mount the disk by reading these instructions, will only find obfuscated data that is not recognized, leaving the disk useless.
- Furthermore, if we want to boot from a software-encrypted disk, the pre-boot kernel must also be left unencrypted. The reason for this is that the blocks where the pre-boot kernel is stored, must be available before the operating system can boot, also meaning that the key has to be available before there is a user interface to ask for a password (called the boot key problem). However, one way to mitigate the vulnerabilities that this presents is to make the operating system hash the pre-boot kernel after boot and compare it against system variables to verify its integrity.

Hardware-based encryption avoids these limitations by transparently encrypting and decrypting every bit at a hardware-level, leaving the software completely unaware that there even exists any encryption.

However, although software-based encryption is not able to encrypt absolutely all the sectors on a disk, it still keeps the fundamental advantage of being able to protect user data even when the operating system is not active. For example, if the data is read directly from the disk when used as an auxiliary drive, the adversary will still only find data undistinguishable from random bits – given that the cipher mode used do not have any weaknesses that can be exploited.

We will not take on the task to debate all pros and cons of hardware-based and software-based encryption, but rather list a few selected advantages for both:

- Hardware-based encryption provides slightly better security by being able to encrypt *every* single bit on the disk, and because the keys used for encryption and decryption is not kept in the computers memory. The latter property avoids attacks like cold boot attack [HSH+08].
- Hardware-based encryption implementations have for along time been using keys as small as 40 bits; even though this is a key length that is considered too short [oSN07]. But luckily, this practice has changed as vendors have started implementing the industry standard cipher, AES, in conjunction with cipher modes especially designed for disk encryption.
- Hardware-based encryption has traditionally been considered to achieve better performance than software-based, as the computer's processing power is not used for the ciphering processes (stealing resources from the end-user), but rather a built-in computational device dedicated for this purpose.
- Software-based encryption is starting to gain ground on the performance area, with the proliferation of multi-core systems. This is good for software-based encryption because disk encryption is usually very parallelizable. The more cores a CPU gets, the faster software encryption will be – if implemented correctly. Nonetheless, modern computers usually have plenty of extra computational power available, which can be used for the ciphering process.
- Software-based encryption has the advantage of being able to be applied to any type of storage media, while the selection of disks employing hardware-based encryption is sparse.
- Software-based encryption has the advantage of cost, as software is cheaper than hardware. Since the marginal cost[8] of software is close to zero, a software-based encryption solution will almost likely always be less expensive than an equivalent hardware-based solution.

In the future, hardware-based encryption drives will probably get better, but so will the services that are available on the core computer platform itself. The

[8]Marginal cost is the change in total cost that arises when the quantity produced changes by one unit, i.e. the cost of producing one more unit of a good.

exponential gain described by Moore's Law has so far benefitted CPUs and its subsystems faster and greater than it has for disk drives themselves. If the exponential gain continues, it will probably continue to affect the general-purpose systems quicker and more effectively than the special-purpose systems like hardware-based encryption solutions. But, as we have only superficially touched upon this topic, and therefore do not want to make any bold statements, we leave this section with saying that only the future will show if the duel between hardware-based and software-based encryption will continue, or if one will supersede the other.

Section 2.4 will describe the software-based disk encryption applications which at the time of writing feature XTS-AES encryption.

### 2.2.2 Narrow-block versus Wide-block Encryption

The second property that distinguishes different disk encryption types is narrow-block encryption and wide-block encryption.

Narrow-block algorithms, as the name implies, operate on small blocks of data. Although there is no universally standardized size, the block size used by narrow-block algorithms is usually 16 bytes. Wide-block algorithms on the other hand encrypt or decrypt a whole sector at the time, i.e. 512 bytes – the most common sector size[9]. Figure 2.13 depicts the difference between a block and a sector.

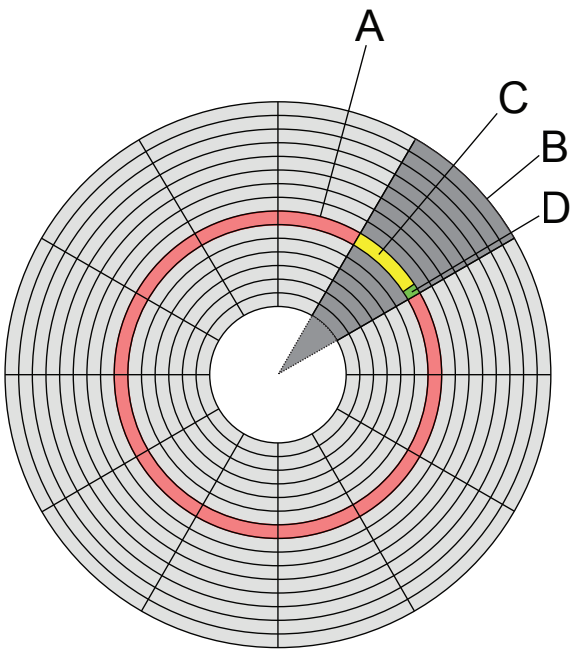



**Figure 2.13:** *The figure shows the difference between a (A) disk track, (B) geometrical sector, (C) disk sector, and (D) disk block.*

[9]In the near future, its expected that the standard sector size will grow up to 1024, 2048, 4096 or even 8192 bytes [Fer06].

Narrow-block algorithms have the advantage of being more efficient, especially when considering hardware implementations [19]. The tradeoff is however that the smaller block size of the narrow-block algorithms provides finer granularity for certain type of attacks [CC07]. Wide-block algorithms are considered to be slow, as they have to process the data through multiple passes [25]. In applications that require streaming reads and writes, the delay or buffer requirements of most wide-block implementations can be prohibitive. But on the plus side, if an implementation has extra available buffer space, computing power, and latency tolerance, wide-block encryption provides better protection than narrow-block encryption [25]. Hence, this is a tradeoff that must be considered depending on the application area.

The P1619 Task Group selected the narrow-block mode XTS for IEEE Std 1619, prioritizing the added efficiency over the additional risk [CC07]. But, recognizing that wide-block encryption ultimately provides better security, P1619.2 was started as a separate project to study wide-block encryption [17].

As convention from this point on, we will use the term narrow-block when referring to a single data block of 16-byte, and the term sector or wide-block when referring to a 512-byte data block.

### 2.2.3 Transparent versus Authenticated Encryption

If the purpose of the encryption is only to protect the privacy of the data this is usually solved using a cryptographic cipher mode for confidentiality. If we also want to guarantee data integrity, this can be achieved by the use of authenticated encryption, which leads us to the last set of properties that distinguishes different types of disk encryption, namely whether the encryption transform is transparent or authenticated.

Transparent encryption is encryption that one is able to place into an existing data path without having to change the data layout or message formats of other components in the data paths [CC07]. This means that the encryption process can be implemented to occur in software, along the data path from the application layer to the storage device, or inside the storage device itself, all without having to modify the data transmission protocols or the data layout on the media.

Authenticated encryption on the other hand, is a type of encryption best suited for tape drives and not sector-level storage devices like hard disks and flash disks. The reason for this is that authenticated encryption produces an authentication tag with every ciphertext (or at least for every set of ciphertexts) to ensure data integrity. This conflict with a fundamental requirement for sector-level storage devices set by the way most disks store information. Disks normally store information in fixed-size sectors, which in turn are written to in a random order. These properties impose two constraints.

The first constraint is that encryption must be performed on a per-data unit basis – either a per-block basis or per-sector basis. The reason for requiring that encryption and decryption is done on a per-data unit basis is simply that

the encryption or decryption of one data unit cannot depend on any other data unit. Consider the following scenario, where we suppose the encryption algorithm works in data units that is dependent on an arbitrary number of other data units. To write data unit $x$, the system first have to read all the other data units that $x$ is dependent on, decrypt them, and then encrypt all of them when the data unit $x$ again is ready to be written back to the disk. Not only is this approach complex, but would also result in very low performance utilization. Furthermore, there are applications, such as databases, that rely on the fact that they can write to data unit $x$ without danger of damaging data unit $x-1$ or $x+1$ [Fer06]. They use this property to ensure that no information (other than possibly the data unit that is being written) is lost in case of crash or power failure.

The second constraint is that the ciphertext cannot be larger than the plaintext. Although highly convenient with respect to data integrity, there is practically no extra room to store additional data, i.e. more room exists but it is infeasible to use it for providing data integrity [Fer06]. Thus, we cannot store message authentication code (MAC) values with the ciphertext. The rationale behind this constraint follows. We could map a 512 byte operating system sector into a 1024 byte disk-sector, but that would result in the loss of half the disk capacity, a price the average user will probably not be willing to pay. We could reserve one in every 16 data unit to store MAC values for the other 15 data units, but this has several problems. First of all, writing to data unit $x$ means updating an additional data unit that contains MAC. This turns a write operation into a read-then-write operation with the associated performance loss. Furthermore, it could damage the MAC data unit (e.g., if there is a power failure), which would lead to the loss of the other 14 data units; also unacceptable. Finally, for various usability, manageability, and deployment reasons, it should be possible to enable and disable disk encryption.

Thus, we see that adding dedicated MAC data units modifies the disk layout and reduces the amount of available disk space in such a way that average users would not accept. Thus, although it is possible to use sector-level storage devices like hard disks and flash disks for authenticated encryption, it is highly unpractical. This leaves authenticated encryption best suited for tape drives used for archive and backup.

To summarize this subsection, we re-capture the two requirements for all transparent encryption modes for block-oriented storage devices:

- The encryption transform must be applicable to individual data-units independently of other data-units and in arbitrary order. This is called the *random-access property*, and implies that no chaining between different data-units is possible.
- The encryption transform must be *length-preserving*, i.e. the length of the ciphertext must be equal to that of the plaintext. This means that the transform must be deterministic, and that it cannot store any nonce, IV, or message authentication code (MAC) value with the ciphertext.

## 2.3 Modes of Operation for Disk Encryption

Section 2.1.4 described two general-purpose cipher modes that can be used for disk encryption when slightly modified. However, there also exist cipher modes that are especially designed for disk encryption. The following paragraphs will describe the most commonly known disk encryption modes with the exception of the XTS mode, as a detailed description is given in chapter 3.

Please note that the following modes only describe encryption of data that is an integral number of either 16-byte or 512-byte blocks (i.e. one can say that they are used in a modified ECB mode even though they are modes of operations themselves). In other words these techniques do not address how to handle data that is not evenly dividable into blocks. This means that in order to use any of the following algorithms in an implementation of disk encryption were the latter is required (e.g., disks using 520-byte sectors), one must address this issue by also choosing a length-preservation algorithm, e.g., ciphertext stealing.

### 2.3.1 LRW: Liskov, Rivest, Wagner

LRW is a tweakable narrow-block cipher mode, short for Liskov, Rivest, Wagner – the trio who inspired to this mode with their paper "Tweakable Block Cipher" [LRW02]. This paper outlined the difference between a conventional block cipher and a tweakable block cipher, and introduced a generic construction for the latter; which became the basis of the LRW construction. LRW with AES as the underlying cipher was for a long time considered the most promising candidate for P1619 [CC04].

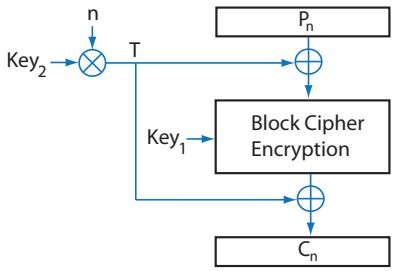



**Figure 2.14:** *LRW mode encryption*

As we recall from 2.1.2, the characteristic feature of a tweakable block cipher mode is the use of a tweak value as an additional input to the ciphering process. The tweak used by LRW is made up by the multiplication between a secret key and the logical index of the data block being encrypted ($T = Key_2 \otimes n$, where $Key_2$ is the secret key, and $n$ is the logical index). The key used for this purpose is not to be derived from the key material of the block cipher (which uses another key - $Key_1$), but has to be supplied additionally [CC04]. By making the tweak dependent of the logical index, LRW effectively ties the ciphertext to a disk location, which in turn mitigates threats such as copy-and-paste attacks and malleability attacks (described in section 4.3).

However, the cost of the modular multiplication, which is performed in GF($2^{128}$) modulo $x^{128}+x^7+x^2+x+1$, is a somewhat slower cipher, due to the complexity of implementing general multiplication [19]. But, an efficient implementation of LRW can make use of the fact that consecutive blocks having the same sector address (i.e. logical index) is able to reuse previously known tweaks (one can easily verify that $T_{i+1} = Key_2 \otimes (i+1) = (Key_2 \otimes i) \oplus (Key_2 \otimes n) = T_i \oplus (Key_2 \oplus n)$). Thus, one might create a precomputed table of values, making it possible to compute the tweak for a full 512-byte sector, for the cost of only 1 multiplication operation, 32 additions and a few integer increments [CC04].

### 2.3.2 XEX: XOR-Encryption-XOR

XEX is another tweakable narrow-block cipher mode, proposed by Rogaway in his paper "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC" [Rog04]. XEX is actually a subroutine in Rogaway's Offset CodeBook (OCB) mode, which is a combined mode for confidentiality and authentication. XEX in it self is a general-purpose algorithm, but when used for disk encryption facilitates efficient processing of consecutive blocks [19].

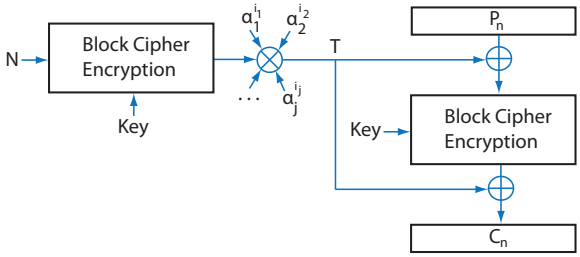



**Figure 2.15:** *XEX mode encryption*

The main construction of XEX is simple, XOR-Encrypt-XOR, but the composition of the tweak is slightly more complex. The tweak, as defined by [Rog04], is comprised of the encrypted value of an index multiplied with several other indexes ($T = E_k(N) \otimes \alpha_1^{i_1} \otimes \alpha_2^{i_2} \otimes \cdots \otimes \alpha_j^{i_j}$). When used for disk encryption the tweak is represented as a multiplication between the sector address and two to the power of the block index inside the sector ($T = E_k(N) \otimes 2^i$, where $N$ is the sector address and $i$ is the block index) [19]. There is some similarity between LRW and XEX, but it is worth emphasizing that while LRW is a cipher mode dedicated to disk encryption, XEX is considered a general block cipher construction which can be modified especially for disk encryption [Rog04].

XEX, when used for disk encryption, is considered not significantly less efficient than LRW [19]. XEX avoids the inconvenience of implementing general multiplication in GF($2^{128}$), but still have to facilitate multiplication in the binary field GF(2) modulo $x^{128} + x^7 + x^2 + x + 1$. On the other hand, XEX requires two cipher calls for each encrypted block; one to encrypt the sector address and one to encrypt or decrypt the actual data block. However, an efficient implementation of XEX can make use of the fact that for consecutive blocks having the same sector address, the encryption of the sector address only has be done once and can be reused for that whole sector afterwards. Moreover, XEX avoids

the time-consuming process of generating multiple keys, since it uses the same key for encrypting the sector address as it does for encrypting or decrypting the actual data block. The security proof of XEX is given in Rogaway's paper [Rog04].

### 2.3.3 MCB: Masked CodeBook

MCB is yet another tweakable narrow-block cipher. This block cipher mode was proposed by El-Fotouh et al. in their paper "A New Block Mode of Operation for Disk Encryption" [EFD08b] in which they proclaim it to be very fast compared to other similar ciphers [EFD08b].

The distinguishing property of MCB is that it uses three keys. The first key (called the *MKey*) is used to generate a mask array – used to further variate the output of each block cipher operation. The second key is used to encrypt the tweak (called the *TKey*). Lastly, the third key is used for encryption or decryption of the actual data block (called the *EKey*). The mask matrix used by MCB contains enough masking data to be used with a whole sector at the time, without having to use the same mask twice. This mask is constructed using AES in counter mode [EFD08b].

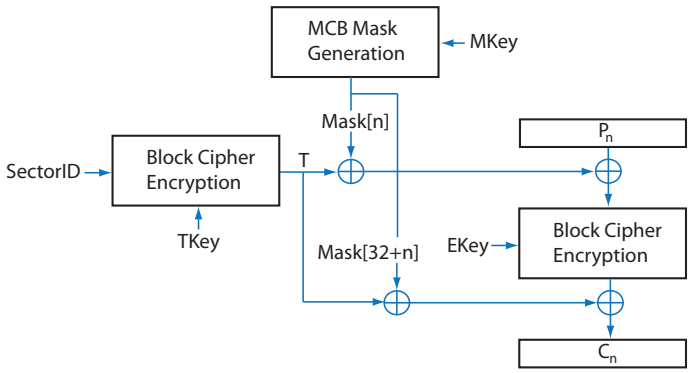



**Figure 2.16:** *MCB mode encryption*

As seen from Figure 2.16, the encryption process of MCB can be divided into three stages – one for each key. The first stage is to compute the tweak by encrypting the sector ID using the second key ($T = E_{TKey}(SectorID)$). The next stage is to produce the masking values (denoted Mask[n] and Mask[32+n] in Figure 2.16), which are derived by XORing the tweak from the previous step with their respective row in the mask matrix. Then the last stage is the same as for XEX, namely XOR-Encrypt-XOR.

Thus, we observe that the MCB mode does not use any modular multiplication but rather a few extra XOR operations to variate the values used in the pre- and post-whitening processes. The authors of MCB argue that the MCB is secure as long as its three keys are not known to the adversary, and have thoroughly discussed the consequences of an adversary knowing one or two of the three keys in [EFD08b].

### 2.3.4 CMC: CBC-Mask-CBC

CMC is a tweakable wide-block cipher mode drafted by Halevi and Rogaway. This cipher mode uses the well known CBC mode in cascade, with a masking stage in between. After the first CBC processing, a mask is computed from the resulting ciphertext, which in turn is applied to all intermediate cipher blocks. This step causes interdependency among the cipher blocks. A second CBC processing is then performed, but now traversing the intermediate ciphertext in reversed order. As in normal CBC, this scheme also uses an IV, but the authors call it a tweak value. The tweak value serves as IV for both CBC steps, making the cipher mode a tweakable cipher mode.

CMC's main advantage is that, since it uses two steps of CBC processing, it is able to reuse an existing implementation of CBC. But, as a consequence of the CBC steps, CMC is not parallelizable. The security proof for CMC is found in Halevi and Rogaway's paper "A Tweakable Enciphering Mode" [HR03b].

### 2.3.5 EME: ECB-Mix-ECB

EME is another tweakable wide-block cipher mode developed by the duo Halevi and Rogaway. In contrast to CMC, EME is parallelizable and thus suitable as cipher mode for high speed storage devices [HR03a]. EME is based on a tweakable enciphering scheme described in the paper "A Parallelizable Enciphering Mode" [HR03a]. As the name suggests, EME uses the ECB mode as a subroutine, applying ECB mode encryption to the plaintext, followed by a mixing step, and then repeating another ECB mode encryption. The cipher mode structure is symmetric, and thus decryption is done the same way.

EME is considered as to having a two main advantages over CMC beyond only it's parallelizability; that all block cipher calls in EME is done using one single key, instead of two; and that enciphering under EME only uses the forward direction of the block cipher, while deciphering only uses the backwards direction. These changes from CMC are convenient when using a cipher such as AES, where the two directions are substantially different [HR03a]. The security proof of EME is based on the assumption that the underlying block cipher used is secure [HR03a].

### 2.3.6 XCB: Extended CodeBook

XCB is yet another wide-block cipher mode, which utilizes a Luby-Rackoff structure [MF04]. This mode was developed by McGrew and Fluhrer of Cisco Systems, Inc. According to the authors, XCB's application area is not only disk encryption, but also other areas where systems cannot allow data expansion, such as some network protocols. The XCB mode is based on a five-round Luby-Rackoff cipher in which the first and last rounds do not use the conventional Feistel structure, but makes use of a single block cipher invocation instead [MF04]. A fundamental drawback of XCB, EME, and CMC in their original form is that they are patented [20].

## 2.4 Disk Encryption Software

There is a vast amount of disk encryption software available, mainly dividable between two categories: commercial and open-source software. In addition, there is also software commercially bundled with operating systems like; BitLocker [23] for Windows Vista and FileVault [1] for Mac OS X v10.3 and later. The most popular open-source disk encryption software includes BestCrypt [5], CrossCrypt [10], dm-crypt [9], DriveCrypt [13], FreeOTFE [11], and TrueCrypt [45]. In the commercial category the list is even longer.

The following paragraphs will describe the open-source cryptographic software which, at the time of writing, features disk encryption using AES in XTS mode. But first, for the sake of simplicity and later discussion, we define three classes of disk encryption software.

### Full disk encryption (1)

Encryption software employing *full disk encryption* encrypts every single bit of data on a disk or partition. The only sector which is not encrypted is the MBR. This sector contains machine code instructions necessary to mount the disk, and thus have to be left unencrypted. This encryption type is suitable for non-system disks when using software-based encryption solutions.

### Full system disk encryption (2)

As mentioned above, the term full disk encryption is used when every bit of data on a disk is encrypted. However, if every single bit of data on a disk is encrypted, it will not be possible to boot from that disk (even though the MBR is left unencrypted). Encryption software employing *full system disk encryption* thus encrypts every bit of data, except the pre-boot kernel and MBR. This encryption type is suitable for system disks (i.e. disks containing the operating system) when using software-based encryption solutions.

### Virtual encrypted partition (3)

Encryption software supporting *virtual encrypted partitions* features standalone file containers that can be mounted as disks or read/written using any other method. These file containers are normal files that can be moved, renamed, and deleted, the same way as other files. Although this is not disk encryption per say, it is included since most disk encryption software also includes this functionality.

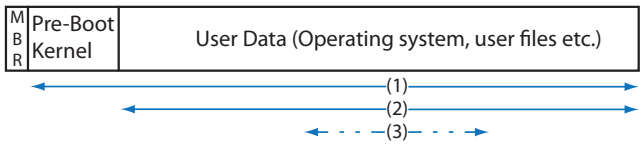



**Figure 2.17:** *The figure illustrates the difference between (1) full disk encryption, (2) full system disk encryption, and (3) virtual encrypted partitions.*

### 2.4.1 TrueCrypt

TrueCrypt [45] is free open-source disk encryption software for Windows, Mac OS X, and Linux. It features strong 256-bit encryption using XTS mode with AES, Twofish, or Serpent block cipher, or all three of them in cascade mode. As of version 5.1, the Windows version of TrueCrypt supports *full system disk encryption*, as well as *full disk encryption* and file containers that may be mounted as *virtual encrypted partitions*.

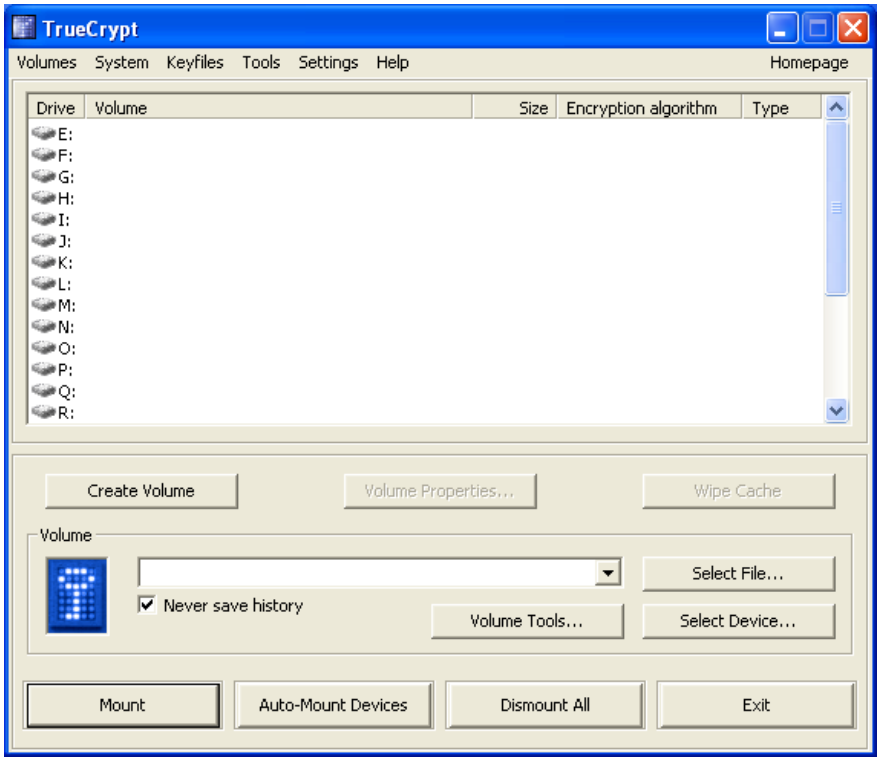



**Figure 2.18:** *TrueCrypt main window*

Once disk encryption is applied, or a virtual encrypted partition is mounted, the encryption/decryption is entirely transparent, and except from a small TrueCrypt icon in task bar there is no visual sign of encryption. If full *system* disk encryption is applied, the user is met with an authentication screen at boot. As of version 6.1, it is even possible to prevent an adversary that is watching you start your computer from knowing that TrueCrypt is in use by turning off all text in the pre-boot authentication screen. This way, when you start the computer, no texts will be displayed by the TrueCrypt boot loader (not even if you enter the wrong password). The computer will appear to be "frozen" while you can type your password [47]. Furthermore, as of version 6.1 it is also possible to create a separate Hidden Operating System [47].

Virtual encrypted partitions exists as normal files on the host filesystem, and must be opened and mounted by TrueCrypt, either by using the main window (depicted in Figure 2.18) or shell commands.

In addition to its encryption features, TrueCrypt provides two levels of plausible deniability, in case an adversary forces you to reveal the password: by the use of hidden volume[10] and the fact that no TrueCrypt volume can be identified (i.e. cannot be distinguished from random data) [47]. One weakness with this scheme is that this feature is not likely to hide the volumes from a seasoned adversary, as there will be disk space that cannot be accounted for, unless they are sufficiently small.

### 2.4.2 FreeOTFE

FreeOTFE [11] is another transparent disk encryption program. It supports both PCs (running Windows 2000/XP/Vista) and PDAs (running Windows Mobile 2003/2005/6). Block ciphers supported by FreeOTFE include AES, Blowfish, RC6, Serpent, Twofish, and others [12]. Unlike TrueCrypt, which supports *full system disk encryption*, FreeOTFE only supports *full disk encryption* and *virtual encrypted partitions*.

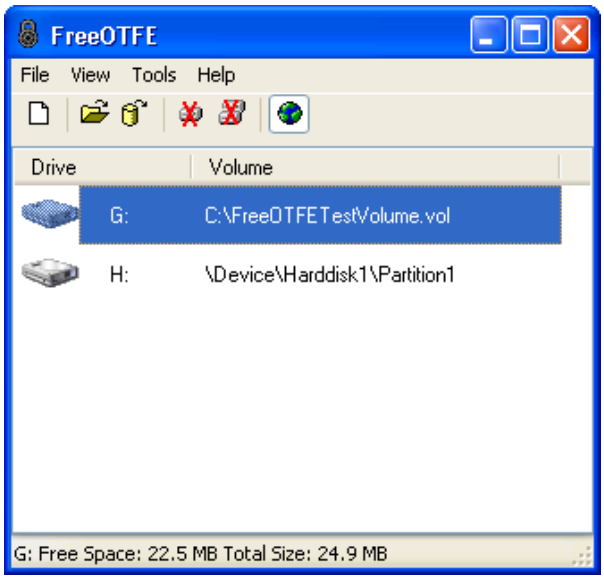



**Figure 2.19:** The FreeOTFE window showing a virtual encrypted partition (G:) and a disk which is full disk encrypted (H:).

The characteristic feature about FreeOTFE is that it provides a modular architecture allowing third parties to implement additional algorithms if required. Originally, FreeOTFE only offered encryption using CBC with ESSIV, but as of version 3.0, the LRW and XTS modes were also introduced [12].

Similar to TrueCrypt, FreeOTFE also allows any number of hidden volumes to be created, providing both plausible deniability and deniable encryption.

[10]Known as steganography, which is the art and science of writing hidden messages in such a way that no-one apart from the sender and intended recipient even realizes there is a hidden message, a form of security through obscurity [Kah97]. By contrast, cryptography obscures the meaning of a message, but it does not conceal the fact that there is a message.

### 2.4.3 dm-crypt

dm-crypt [9] is included in Linux kernel versions 2.6 and later, and is able to perform *full system disk encryption*, *full disk encryption*, make *virtual encrypted partitions*, and encrypt specific files. dm-crypt uses cryptographic routines from the UNIX kernel's Crypto API [24], supporting modes of operations like XTS, LRW, and CBC with ESSIV.

Used with initrd[11], dm-crypt is also able to provide a pre-boot authentication mechanism similar to TrueCrypt. Furthermore, dm-crypt encrypted disks can be accessed and used under Windows using FreeOTFE, provided that the filesystem used is supported by Windows (i.e. FAT/FAT32/NTFS). ext3 and ext2 filesystems can also be mounted using the ext2 Installable File System driver for Windows [41].

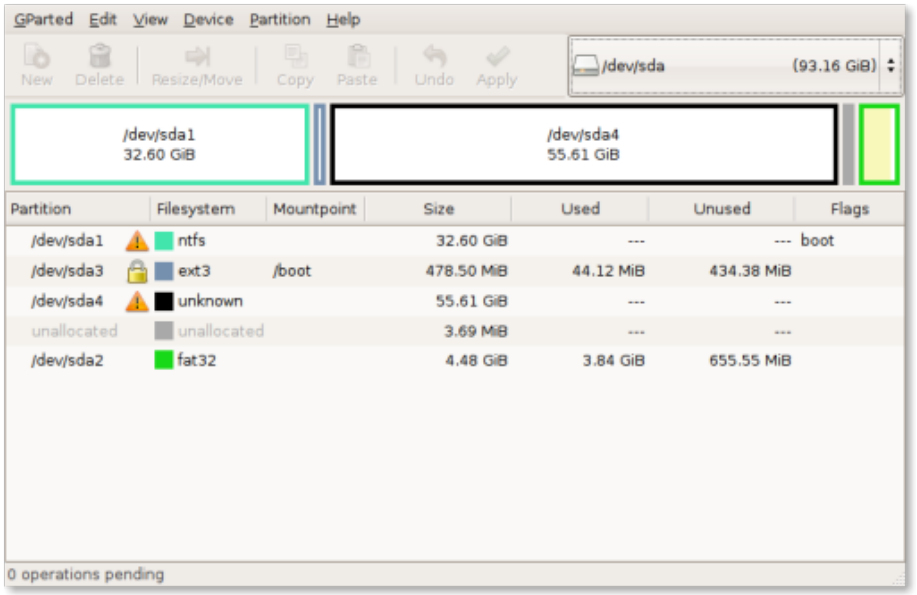



**Figure 2.20:** *The drive map in the partition software GParted show that the dm-crypt partition (/dev/sda4) is unrecognized, while the other LVM volumes is recognized as distinct filesystems.*

The dm-crypt device mapper target resides in the kernel space of Linux, and is concerned only with encryption and decryption – it does not interpret any data itself. It relies on user space front-ends to create and activate encrypted volumes, and manage authentication. At least two frontends are currently available: *cryptsetup* [9] and *cryptmount* [8].

[11] initrd or the initial ramdisk is a temporary file system commonly used by the Linux kernel during boot. The initrd is typically used for making preparations before the real root file system can be mounted.

# Chapter 3

# IEEE Std 1619-2007

*The nice thing about standards is that there are so many of them to choose from.*

**Andrew S. Tanenbaum**

IEEE Std 1619 [CC07] is the most recent IEEE standard regarding the encryption of data on block-oriented storage devices. It was approved by the IEEE-SA Standard Board in December 2007 [2] and published in April 2008.

The purpose of this chapter is give a complete and detailed description of IEEE Std 1619-2007; that describes the XTS-AES algorithm and a standard for XML-based key export format. First, we will briefly skim the history of this standard describing the various solutions considered. Next, we will describe the scope of the standard, comparing it with the other family members of Project 1619. Then, after shortly discussing related work we will elaborate on the XTS-AES algorithm by first describing the general XTS mode and then thoroughly explaining XTS-AES and its encryption and decryption procedures. Lastly, we will address the XML-based key export format.

## 3.1 History

IEEE Std 1619 started out as Project 1619 in August 2002 after the formation of IEEE Computer Society's *Security in Storage Working Group (SISWG)*. Originally, the objective of the project was to create *one* encryption standard suitable for data storage devices [HC03]. But starting from mid 2005, the objective of P1619 was divided up into several sub projects (see Figure 3.1), each with its own scope, whereas P1619's became narrow-block encryption.

Before the P1619 were split into several sub projects, the first mode of operation which was considered was the wide-block EME mode, created by Shai Halevi and Phil Rogaway in 2003. The EME transform was proposed to be used with AES as the underlying cipher, acting on data blocks of 512 bytes. The draft proposal mode was called EME-32-AES, and is described in [CC03]. This mode was however proven vulnerable by Antione Joux in his paper "Cryptanalysis of

the EMD Mode of Operation"[1] [Jou03], which was presented at EuroCrypt 2003.

This led to the next mode considered by SISWG for P1619, namely the narrow-block LRW mode. LRW was also proposed to be used with AES as the underlying cipher, but would operate on data blocks of 128 bits. The draft proposal mode was called LRW-AES, and is described in [CC04]. The LRW mode was suggested in April 2004, and was until mid 2006 considered the most promising candidate for P1619. During this period, the scope of P1619 also changed to focus merely on narrow block encryption modes. But LRW fell when Niels Ferguson of Microsoft noted in Crypto 2006 that you can leak the tweak key if encrypted with itself [3]. Consequently in August 2006 after a straw poll in SISWG, the P1619 Task Group abandoned LRW [19].

When the focus of P1619 changed to narrow-block encryption modes specifically, Mart Somera pointed SISWG to the XEX mode in January 2006 [3]. But being a general purpose algorithm, the P1619 Task Group had to customize its construction. Without going into details (because those will shortly be described in the following sections) the result was a XEX-based algorithm called XTS-AES, which in turn was chosen to be included in the standard. P1619 was approved in December 2007 and named *IEEE Std 1619: Cryptographic Protection of Data on Block-Oriented Storage Devices*, describing the XTS-AES encryption algorithm and a standard for an XML-based key-export format.

In early 2008, IEEE submitted XTS to NIST for consideration as an encryption mode of operation for the AES block cipher. NIST held a public review period from June 2008 to September 2008 and very recently, Matthew Ball, the Chair of SISWG, announced in an email [27] to the P1619 Task Group that NIST had accepted XTS as an approved mode of operation for AES under FIPS 140[2].

## 3.2 Scope

The scope of IEEE Std 1619 is to define *a standard architecture for cryptographic protection of data on block-oriented storage devices* [CC07]. This architecture consists of two elements, namely XTS-AES and a XML-based key backup structure.

XTS-AES, which is usually considered the main part of the standard, is a transparent narrow-block encryption mode of operation especially designed for disk encryption that can be implemented in both software and hardware. The other main part of the standard is the XML-based key backup structure that aims to facilitate an application independent way of sharing the keys used in encryption and decryption processes. This is achieved by defining the key material and necessary settings in the unified data format XML.

[1] Notice that the article is a cryptanalysis of the Encrypt-Mask-Decrypt (EMD) mode. But EME, which only differs from EMD by not being parallelizable, is also subject to the same attacks [Jou03].

[2] The Federal Information Processing Standard 140 (FIPS) are series of publications numbered 140 which are a U.S. government computer security standards that specify requirements for cryptography modules. As of December 2006, the current version of the standard is FIPS 140−2, issued on 25 May 2001 [36].

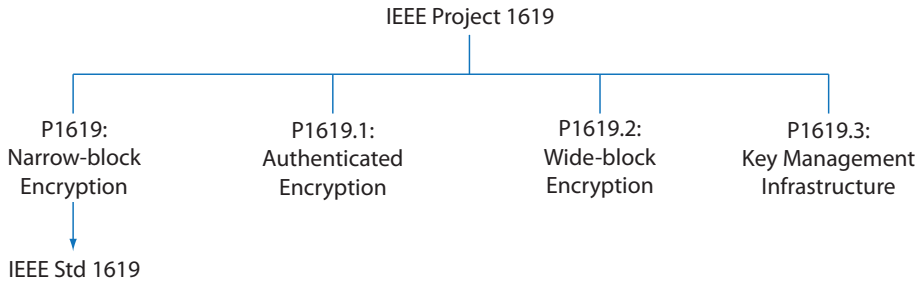



**Figure 3.1:** *The scope of IEEE Std 1619 is to define a standard architecture for cryptographic protection of data on block-oriented storage devices.*

In the future, IEEE Std 1619 will be complemented by P1619.2 and P1619.3. P1619.2 is aimed at defining a standard for wide-block encryption [17]. This will complement IEEE 1619 by providing an alternative storage encryption algorithm for block-oriented storage devices, but with a wide-block granularity. P1619.3 will complement the XML-based key-backup structure defined in IEEE Std 1619 by also defining methods for the storage, management, and distribution of cryptographic keys used for the protection of stored data [18].

As depicted by Figure 3.1, there is also one more family member, namely P1619.1. P1619.1 was approved and published as IEEE Std 1619.1 together with IEEE Std 1619. As the name of the standard suggests, it describes methods and algorithms for authenticated disk encryption [16]. In light of the difference explained in section 2.2.3, we are able to identify it as a standard suitable for tape drives, and not primarily block-oriented storage devices such as hard disks and flash disks, which is the scope of IEEE Std 1619 and P1619.2.

## 3.3 Related Work

To our knowledge, there is no other initiative specifically aimed for standardizing narrow-block encryption. However, IEEE's SISWG is not the only standardization body involved in storage security initiatives in general. In addition to SISWG, the Internet Engineering Task Force (IETF), InterNational Committee for Information Technology Standards (INCITS), International Organization for Standardization/International Electrotechnical Commission Joint Technical Committee (ISO/IEC JTC1), Storage Networking Industry Association (SNIA), Distributed Management Task Force (DMTF), and Trusted Computing Group (TCG) all have projects related to storage security [14].

Of these, the standardization body most often referred to in popular media is TCG, which recently announced release of final versions of their storage specifications named *Opal Security Subsystem Class Specification* for PC clients and *Enterprise Security Subsystem Class Specification* for data center storage [39]. However, these are both specifications for hardware-based encryption solutions, yielding them interesting only to storage device vendors. Moreover, they do not recommend any specific mode of operation other than using the AES cipher [6].

## 3.4 XTS-AES

XEX-based Tweakable CodeBook mode with Ciphertext Stealing (XTS)[3] is the disk encryption mode of operation proposed by the P1619 Task Group. XTS with AES as the underlying block cipher gave birth to the tweakable narrow-block encryption algorithm *XTS-AES*. XTS-AES comes in two flavors: XTS-AES-128 and XTS-AES-256. As the names suggests XTS-AES-128 is the mode using 128-bit keys, and XTS-AES-256 is the mode using 256-bit keys. To be compliant with the standard, an implementation must support at least one of the above modes [CC07].

The XTS cipher mode can be broken down into three parts, namely: the XEX-based construction, tweakable codebook mode, and ciphertext stealing. The following paragraphs will give a very brief and holistic description of the XTS mode, while later sections will describe the XTS-AES cipher mode in detail.

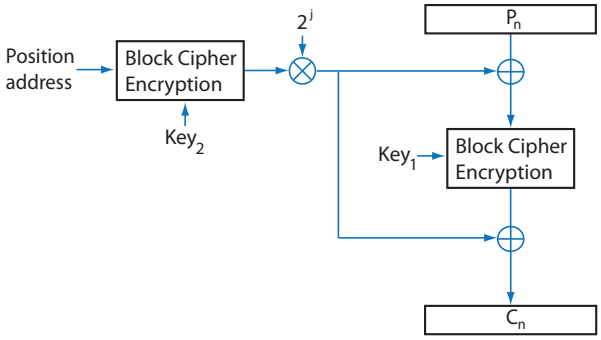



**Figure 3.2:** *XTS mode encryption*

The XEX-based construction means that XTS follows the same steps as XEX, when the general-purpose algorithm XEX is modified for disk encryption (see Figure 3.2). One important modification from the XEX design is however the use of two keys, instead of one. The first key ($Key_1$) is needed in the encryption step of XOR-Encrypt-XOR. While the second key ($Key_2$) is used to encrypt the position address value.

The term tweakable codebook mode (TCB) comes from the fact that XTS uses electronic codebook mode (ECB), but with two additional inputs in each ciphering step, hence the name tweakable. Although the ECB based nature of XTS is not clearly illustrated in the figure above because it only depicts the encryption of a single data block, the ECB similarity of TCB will become clear when encryption and decryption of multiple data blocks is described later in this chapter.

Lastly, in order to adapt the algorithm to also be able to process data that is not dividable into even 128-bit blocks, XTS makes use of the ciphertext stealing (CTS) technique when needed.

[3] Although XEX-TCB-CTS should be abbreviated as XTC, "C" was replaced with "S" (for "stealing") to avoid confusion with the abbreviation for the ecstasy drug, which is also XTC.

Before continuing with a detailed description of how XTS-AES encryption (3.4.1) and decryption (3.4.2) is performed, the next couple of paragraphs will define some key terms used to describe XTS-AES.

**Data block.** In data storage, a block is a sequence of bits or bytes, having a nominal length (i.e. a block size). The XTS-AES algorithm refers to a data block as the single block of plaintext or ciphertext of 128 bits in size. According to the standard, each data block is assigned a non-negative block number [CC07].

**Data unit.** A data unit on the other hand is the sequence of one or more data blocks. Figure 3.3 depicts such a sequence of data blocks, comprising a data unit. According to the standard, each data unit must be assigned a position address (logical address or sector address) that is a non-negative integer [CC07].

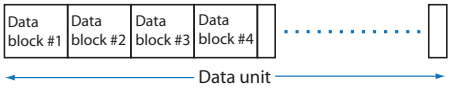



**Figure 3.3:** *The number of 128-bit blocks in a data unit shall not exceed* $2^{128} - 2$ *[CC07].*

The mapping between the data unit and the transfer, placement, and composition of data on the storage device is however not defined by IEEE Std 1619. But, in order to be compliant with the standard, an implementation must include documentation describing this mapping [CC07]. Thus, an encrypted data unit does not necessarily have to correspond to a physical block address on the storage device, hence the generic term position address.

**Tweak.** As described in section 2.1.2, a tweak is an extra input used to further variate the output of a cryptographic function. The tweak used in XTS-AES is a 128-bit value, represented by the multiplication between the encrypted value of the position address (either a logical address or sector address) of the data, with two (2) to the power of the block number inside the data unit.

**Key scope.** Key scope is defined as a range of data encrypted with the same XTS-AES key. The key scope is identified by three non-negative integers:

- The position address corresponding to the first data unit
- The size in bits of each data unit
- The number of units to be encrypted/decrypted under the control of this key

An XTS-AES key shall not be associated with more than one key scope. The reason for this restriction is that encrypting more than one block with the same key and same index introduces vulnerabilities that might potentially be used in an attack on the system [CC07].

**Modular multiplication.** $\otimes$ is used in the following figures to denote modular multiplication of two polynomials over the binary field GF(2), modulo $x^{128} + x^7 + x^2 + x + 1$. Details about how modular multiplication is performed can be found in [MvOV01, Sta06].

### 3.4.1 XTS-AES Encryption Procedure

#### Encryption of a Single Data Block

The most elementary building block of the XTS-AES algorithm (after the AES cipher, of course) is the scheme for encrypting a single 128-bit block of plaintext. In IEEE Std 1619, this is denoted *XTS-AES-blockEnc* and expressed mathematically in Equation 3.1, and graphically in Figure 3.4.

$$C \leftarrow \text{XTS-AES-blockEnc}(Key, P, i, j) \tag{3.1}$$

where

- $Key$ is the 256 or 512 bit XTS-AES key, which is composed by two fields of equal size, namely $Key_1$ and $Key_2$, such that $Key = Key_1 \mid Key_2$.
- $P$ is a 128-bit block of plaintext
- $i$ is a 128-bit value representing the position address
- $j$ is the sequential number of the 128-bit block inside the data unit
- $C$ is the 128-bit block of ciphertext resulting from this operation

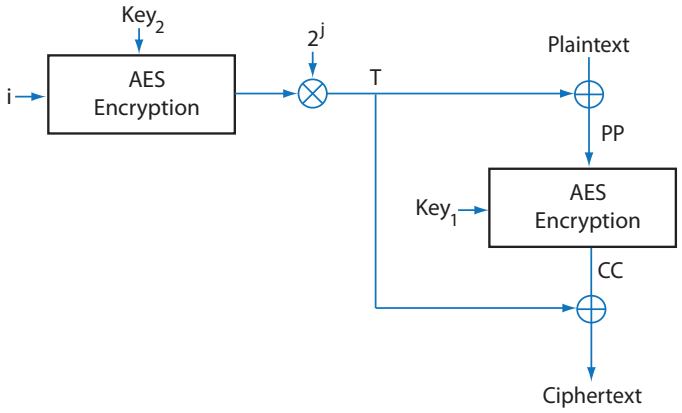



**Figure 3.4:** *XTS-AES-blockEnc*

As seen from Equation 3.1, *XTS-AES-blockEnc* takes four inputs; the key set, 128 bits of plaintext, the position address, and a block number; which is used to produce 128 bits of ciphertext. The encryption procedure of XTS-AES for a single 128-bit data block follows. First, the position address ($i$) is encrypted using $Key_2$ and multiplied with $2^j$. Two (2) represents the second element of GF($2^{128}$) that corresponds to the polynomial $x$ (i.e. $0x2_{16}$ in hexadecimal, and $0000...0010_2$ in binary), and $j$ is the sequential number of the 128-bit block inside the data unit being processed.

The result of the modular multiplication is the Tweak ($T$), which is used twice; first to be XORed with the plaintext, resulting in $PP$; and secondly XORed with $CC$ (the result of encrypting $PP$ with $Key_1$), which gives the ciphertext.

### Encryption of a Data Unit

As data can be of variable length, Equation 3.2 and Figure 3.5 describes the encryption procedure for a data unit consisting of multiple data blocks.

$$C \leftarrow \text{XTS-AES-Enc}(Key, P, i) \tag{3.2}$$

where $Key$, $P$, $i$ and $C$ is defined the same way as for Equation 3.1.

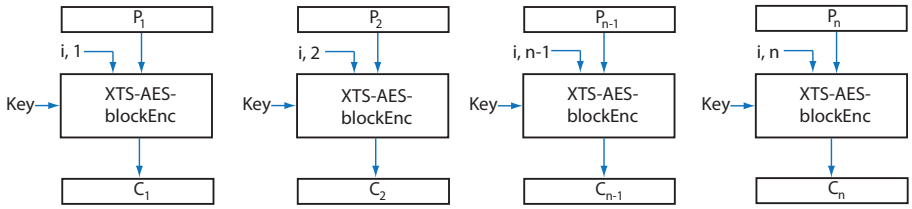



**Figure 3.5:** *XTS-AES-Enc used on plaintext dividable in 128-bit blocks. Notice the use of XTS-AES-blockEnc as a subroutine.*

The plaintext is first partitioned into $n$ blocks, $P = P_1 \mid ... \mid P_{n-1} \mid P_n$. If the plaintext size is evenly dividable into $n$ 128-bit blocks, the XTS-AES-Enc procedure is done straight forward in tweakable codebook mode with the position address ($i$) and the block number ($j$) as additional inputs.

On the other hand, if the plaintext size is not dividable in 128-bit blocks, CTS is used. This means that the first *n-2* blocks $P_1, ..., P_{n-2}$ are encrypted as in the latter case (plaintext is dividable in 128-bit blocks), while the second to last block $P_{n-1}$ and last block $P_n$ utilizes the CTS technique depicted in Figure 3.6.

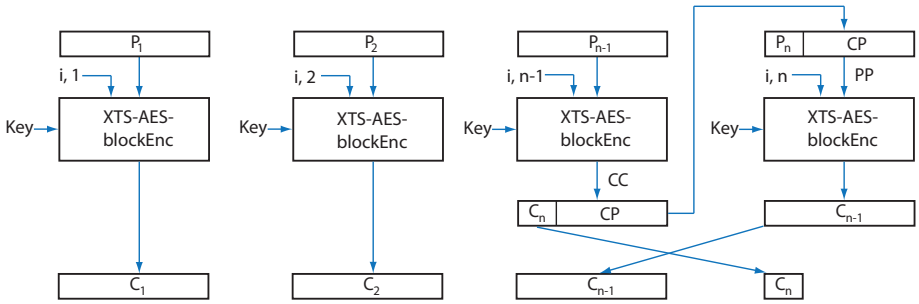



**Figure 3.6:** *XTS-AES-Enc used on plaintext not dividable in 128-bit blocks*

After encrypting the plaintext blocks $P_1$ to $P_{n-2}$ in normal TCB mode, $P_{n-1}$ is also encrypted, but the result ($CC$) is split in to two; $C_n$ which becomes the last block of ciphertext, and $CP$ which is used as input to the last encryption operation ($PP$), along with the last bits of the plaintext $P_n$. This last encryption operation results in the second to last ciphertext, namely $C_{n-1}$.

### 3.4.2 XTS-AES Decryption Procedure

#### Decryption of a Single Data Block

Decryption of a single data block is almost the exact opposite of encryption described in section 3.4.1. In IEEE Std 1619, this is denoted *XTS-AES-blockDec* and expressed mathematically in Equation 3.3, and graphically in Figure 3.7.

$$P \leftarrow \text{XTS-AES-blockDec}(Key, C, i, j) \tag{3.3}$$

where

- $Key$ is the 256 or 512 bit XTS-AES key, which is composed by two fields of equal size, namely $Key_1$ and $Key_2$, such that $Key = Key_1 \mid Key_2$.
- $C$ is a 128-bit block of ciphertext
- $i$ is a 128-bit value representing the position address
- $j$ is the sequential number of the 128-bit block inside the data unit
- $P$ is the 128-bit block of plaintext resulting from this operation

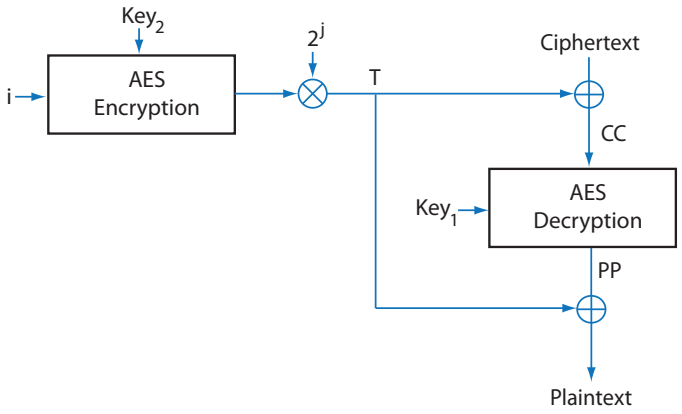



**Figure 3.7:** *XTS-AES-blockDec*

If we were to compare Figure 3.7 with Figure 3.4, we see that the only difference is that while *XTS-AES-blockEnc* makes use of the AES encryption procedure, *XTS-AES-blockDec* uses the reverse procedure (i.e. AES decryption). But, notice that the AES encryption process is still used to derive the tweak $T$, as it is impossible to use the reverse procedure and still get the correct $T$. The decryption procedure of XTS-AES for a single 128-bit data block follows.

Just like for encryption, the position address ($i$) is encrypted using $Key_2$ and multiplied with $2^j$ in GF($2^{128}$). Where two (2) still is the second element of GF($2^{128}$) that corresponds to the polynomial $x$, and $j$ still is the sequential number of the 128-bit block inside the data unit being processed. The result of modular multiplication is ($T$), which is used twice; first to be XORed with the ciphertext, resulting in $CC$; and secondly XORed with $PP$, which in turn gives the plaintext.

### Decryption of a Data Unit

Just like the encryption process of a data unit needs to take into account that data can be of variable length, so must the decryption process. Equation 3.4 and Figure 3.8 describes the decryption procedure for a data unit consisting of a multiple data blocks.

$$P \leftarrow \text{XTS-AES-Dec}(Key, C, i) \tag{3.4}$$

where $Key$, $C$, $i$ and $P$ is defined the same way as for Equation 3.3.

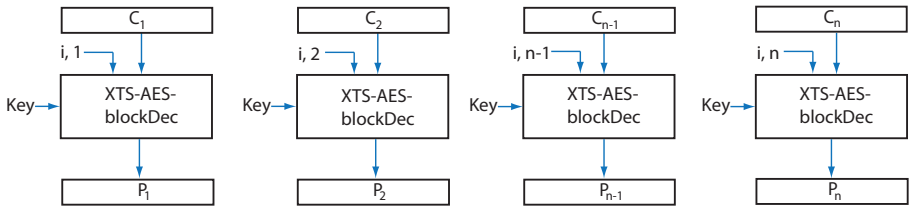



**Figure 3.8:** *XTS-AES-Dec used on plaintext dividable in 128-bit blocks. Notice the use of XTS-AES-blockDec as a subroutine.*

The ciphertext is first partitioned into $n$ blocks, $C = C_1 \mid ... \mid C_{n-1} \mid C_n$. If the ciphertext size is evenly dividable into $n$ 128-bit blocks, the XTS-AES-Dec procedure is done straight forward in tweakable codebook mode using the position address ($i$) and the block number ($j$) as additional inputs.

But if the ciphertext size is not dividable in 128-bit blocks, CTS is used. This means that the first *n-2* blocks $C_1, ..., C_{n-2}$ are decrypted as in the latter case (plaintext is dividable in 128-bit blocks), while the second to last block $C_{n-1}$ and last block $C_n$ utilizes the CTS technique depicted in Figure 3.9.

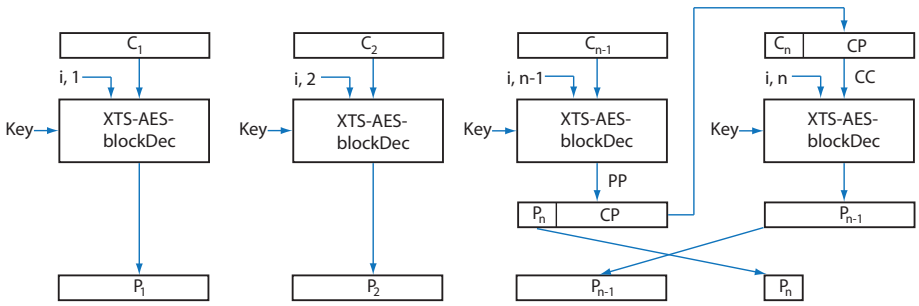



**Figure 3.9:** *XTS-AES-Dec used on plaintext not dividable in 128-bit blocks*

After decrypting the plaintext blocks $C_1$ to $C_{n-2}$ in normal TCB mode, $C_{n-1}$ is also decrypted, but the result ($PP$) is split in to two; $P_n$ which becomes the last block of plaintext, and $CP$ which is used as input to the last decryption operation ($CC$), along with the last bits of the ciphertext $C_n$. This last decryption operation results in the second to last plaintext, namely $P_{n-1}$.

## 3.5 XML-based Key-Export Format

The IEEE 1619 standard also defines a XML-based key-export format. A device compliant with this standard must consequently also support its key backup structure [CC07].

The key backup structure provides all the information that is needed in order to decrypt an arbitrary number of data units that is encrypted with XTS-AES. But before we begin describing the key backup structure, the following paragraphs will clarify and define some terms used later on.

#### XML

Extensible Markup Language (XML) [53] as the well-known general-purpose markup language is used to represent the key backup structure of IEEE Std 1619. Describing the key in such a unified format facilitates application independent ways of sharing key material, which is one the goals of the IEEE Std 1619 [CC07].

#### DTD

While XML is a markup language used to create other markup languages, a Document Type Definition (DTD) [52] is needed to create and describe the language. In our setting, a DTD is used to define the elements that need to be present in the key backup structure.

#### Base64

Base64 refers to a specific content transfer encoding. It is used as a generic term for any similar encoding scheme that encodes binary data by treating it numerically and translating it into a base 64 representation. As we will see later on, some of the content in the key backup structure is encoded with Base64, according to IETF RFC 3548 [21].

Without going into details, Base64 encoding obfuscates the original content into a form that is not human readable. The Base64 encoding used in the key backup structure defined by IEEE 1619 can be compared to substitution ciphering[4]. The substitution table and other detailed information regarding the Base64 encoding can be found in RFC 3548.

[4]Substitution ciphering is a method of encryption by which units of plaintext are replaced with ciphertext according to a regular system or substitution table [Sta06]. The receiver deciphers the text by performing an inverse substitution.

### 3.5.1 Key Backup Structure Overview

The leftmost column of Table 3.1 lists the five elements that need to be present in the key backup structure in order to be compliant with IEEE Std 1619. Each of these is explained further in their respective tables below, while an example of a XML document that fulfills these is shown in 3.5.2. Tables are adapted from [CC07].

| **Element** | **Description** | **Reference** |
|---|---|---|
| *StructureID* | Identifier of current structure | Table 3.2 |
| *Standard* | Standard identifier | Table 3.3 |
| *KeyScope* | Key scope | Table 3.4 |
| *Transform* | Transform description | Table 3.5 |
| *KeyMaterial* | Key material and its length | Table 3.6 |

**Table 3.1:** *Key backup structure*

#### StructureID

| **Element** | **Size** | **XML Encoding** | **Description** |
|---|---|---|---|
| ID | 16 bytes | Base64 | General identifier for the key backup structure. |
| Comment | Up to 1024 bytes | Text | A text description provided by the vendor. |

**Table 3.2:** *StructureID contains the information needed to uniquely identify a particular instance of a key backup structure.*

#### Standard

| **Element** | **Size** | **XML Encoding** | **Description** |
|---|---|---|---|
| StandardNumber | Up to 128 bytes | Text | Number of the standard used. Shall be IEEE STD 1619-2007. |
| StandardComment | Up to 256 bytes | Text | Any additional standard-related information. |

**Table 3.3:** *Standard defines information about the standard to which the data units were encrypted. Shall be set to IEEE STD 1619-2007.*

**KeyScope**

| Element | Size | XML Encoding | Description |
|---|---|---|---|
| KeyScopeStart | 16 bytes | Integer | Value of the tweak associated with the first data unit in the scope. |
| DataUnitSize | 16 bytes | Integer | The number of bits in one data unit that is covered by the current key. |
| KeyScopeLength | 16 bytes | Integer | The number of data units that are covered by the current key. |

**Table 3.4:** *KeyScope defines the scope of the key material that is identified in the key backup structure. The KeyScope is a sequence of data units, numbered consecutively starting at a certain position.*

**Transform**

| Element | Size | XML Encoding | Description |
|---|---|---|---|
| TransformName | Up to 16 bytes | Text | The transform name. |

**Table 3.5:** *The Transform name shall be one of the supported strings, XTS-AES-128 or XTS-AES-256.*

**KeyMaterial**

| Element | Size | XML Encoding | Description |
|---|---|---|---|
| KeyLength | 2 bytes | Integer | Length (in bits) of the key. Allowed values are 256 (for XTS-AES-128) and 512 (for XTS-AES-256). |
| KeyValue | Variable | Base64 | The value of the key. |

**Table 3.6:** *KeyMaterial is defined by the KeyLength and KeyValue.*

### 3.5.2 XML Format

The DTD for key backup format is shown in Figure 3.10, while an example of an XML document following this DTD is shown in Figure 3.11.

```
<!ELEMENT KeyBackup (StructureID, Standard, KeyScope, Transform, KeyMaterial)>
    <!ELEMENT StructureID  (ID, Comment?)>
    <!ELEMENT ID (#PCDATA)>
        <!ATTLIST ID Encoding CDATA #FIXED "Base64">
    <!ELEMENT Comment      (#PCDATA)>
    <!ELEMENT Standard         (StandardNumber, StandardComment?)>
    <!ELEMENT StandardNumber (#PCDATA)>
    <!ELEMENT StandardComment (#PCDATA)>
    <!ELEMENT KeyScope (KeyScopeStart, DataUnitSize, KeyScopeLength)>
    <!ELEMENT KeyScopeStart    (#PCDATA)>
        <!ATTLIST KeyScopeStart Encoding CDATA #FIXED "Integer">
    <!ELEMENT DataUnitSize         (#PCDATA)>
        <!ATTLIST DataUnitSize Encoding CDATA #FIXED "Integer">
    <!ELEMENT KeyScopeLength   (#PCDATA)>
        <!ATTLIST KeyScopeLength Encoding CDATA #FIXED "Integer">
    <!ELEMENT Transform        (TransformName)>
    <!ELEMENT TransformName    (#PCDATA)>
    <!ELEMENT KeyMaterial  (KeyLength, KeyValue)>
    <!ELEMENT KeyLength        (#PCDATA)>
        <!ATTLIST KeyLength Encoding CDATA #FIXED "Integer">
    <!ELEMENT KeyValue             (#PCDATA)>
        <!ATTLIST KeyValue Encoding CDATA #FIXED "Base64">
```

**Figure 3.10:** *DTD for key backup format [CC07]*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE KeyBackup SYSTEM "keybackup.dtd">
<KeyBackup>
    <StructureID>
                <ID Encoding="Base64">YUBlJHJqMDNhWjFAJCVwXQ==</ID>
                <Comment>Comment text here</Comment>
    </StructureID>
    <Standard>
                <StandardNumber>IEEE STD 1619-2007</StandardNumber>
                <StandardComment>Disk</StandardComment>
    </Standard>
    <KeyScope>
                <KeyScopeStart Encoding="Integer">0</KeyScopeStart>
                <DataUnitSize Encoding="Integer">4096</DataUnitSize>
                <KeyScopeLength Encoding="Integer">1083</KeyScopeLength>
    </KeyScope>
    <Transform>
                <TransformName>XTS-AES-256</TransformName>
    </Transform>
    <KeyMaterial>
                <KeyLength Encoding="Integer">512</KeyLength>
                <KeyValue Encoding="Base64">
                            IUApKFQlWEpHJCkoVypUJVgoKU5UJV
                            dYKShXJVhOSlJFR0gpSCgjJWd0eDk3
                            d3h0NW03NTNobXR4ISNkZjRzZw==
                </KeyValue>
    </KeyMaterial>
</KeyBackup>
```

**Figure 3.11:** *An example XML document containing a key for a single key scope [CC07].*

A key backup structure such as the one depicted in Figure 3.11, can further be protected by encrypting the key material part (KeyMaterial defined in Table 3.6). This is done by encrypting the `<KeyValue></KeyValue>` part with XML-ENC [51] and embedded within the XML key backup structure. Any key wrap algorithm[5] allowed by the XML-ENC specification can be used, but it is mandatory to at least support NIST AES-CBC-256 [CC07].

The location of the wrapping key (i.e. the key used to encrypt the XTS-AES key) is not specified in IEEE Std 1619. But, the cryptographic strength of the wrapping keys should be at least equivalent to the strength of the storage encryption keys wrapped. For example, to wrap an XTS-AES-256 key one should at least use AES with 256-bit keys as the wrapping algorithm [CC07]. More about how to choose an appropriate wrap algorithm can be found in NIST Key Management Guidelines [28].

Figure 3.12 shows an example where the KeyMaterial is encrypted using AES-256 Wrap Key, whose wrapping key has the identifier `WrapKey`. This example corresponds to the unwrapped example in Figure 3.11. The Base64 encoding of the wrapping key used in Figure 3.12 is as follows [CC07]:

```
9s7VKp6PYKOXtYjs5OFBoqCDA3MmFd5tTqYnZv+PVro=
```

[5] A key wrap algorithm is encryption algorithms designed to encapsulate (encrypt) cryptographic key material.

```
<KeyBackup>
    <StructureID>
                <ID Encoding="Base64">YUBlJHJqMDNhWjFAJCVwXQ==</ID>
                <Comment>Comment text here</Comment>
    </StructureID>
    <Standard>
                <StandardNumber>IEEE STD 1619-2007</StandardNumber>
                <StandardComment>Disk</StandardComment>
    </Standard>
    <KeyScope>
                <KeyScopeStart Encoding="Integer">0</KeyScopeStart>
                <DataUnitSize Encoding="Integer">4096</DataUnitSize>
                <KeyScopeLength Encoding="Integer">1083</KeyScopeLength>
    </KeyScope>
    <Transform>
                <TransformName>XTS-AES-256</TransformName>
    </Transform>
    <KeyMaterial>
                <KeyLength Encoding="Integer">512</KeyLength>
                <KeyValue Encoding="Base64">
                    <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
                                Type="http://www.w3.org/2001/04/xmlenc#Content">
                          <xenc:EncryptionMethod
                                Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
                                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
                          <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                                <ds:KeyName xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                                            WrapKey
                                </ds:KeyName>
                          </ds:KeyInfo>
                          <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                                <xenc:CipherValue
                                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                                    MluzVD5PGeoneuFP0bgG3o1bzGVRr
                                    CLG5pR00ER/eTyBTRNSEdmOyT3Q/2
                                    ZGdNn4plzIAml5QYgCKjOTJMPWxzZFZH75/S3SHA
                                    haOYhy4DXovhf+LiiXvThqxWcGaIXS6a4+X82vBgT8j2JRqPe/+A==
                                </xenc:CipherValue>
                          </xenc:CipherData>
                    </xenc:EncryptedData>
                </KeyValue>
    </KeyMaterial>
</KeyBackup>
```

**Figure 3.12:** *The same XML document as depicted in Figure 3.11, but with the KeyValue wrapped using AES-256 Key Wrap [CC07].*

# Chapter 4

# Security Assessment

*Treat your password like your toothbrush. Don't let anybody else use it, and get a new one every six months.*

**Clifford Stoll**

This chapter will give a security assessment of XTS-AES. First, we will discuss the computational security of XTS-AES. Secondly, we will provide the reader with a synopsis of the comments set forward by the cryptographic community in response to the request for comments on XTS-AES by NIST. Lastly, we will describe attacks that may be applicable to XTS-AES.

## 4.1 Computational Security

As described in section 2.1.1, a cryptosystem can be considered computationally secure if the complexity of an attack exceeds the computational capability of the adversary. This section will show that XTS-AES provides a security guarantee that makes it the narrow-block cipher of choice for storage devices seeking the cryptographic goal of confidentiality. Since the security of XTS is based upon the security of XEX, we will first consider XEX and then proceed with XTS. For details regarding the mathematical deductions and formal security proofs, please refer to [Rog04, Min07] for XEX and [CC07, LM08] for XTS-AES.

### 4.1.1 General XEX Transform

The security analysis of the general-purpose algorithm XEX is given by Rogaway in [Rog04] and shows that XEX is secure as long as the number of blocks that are encrypted under the same key is sufficiently smaller than the birthday bound value of $2^{\frac{n}{2}}$, where $n$ is the block size of the underlying block cipher.

The security proof is given by assuming an adversary that is able to make arbitrary number of encryption and decryption queries to the tweakable cipher, using arbitrary number of tweak values. These queries are answered by either the XEX construction or by a truly random collection of permutations and

their inverses over $\{0,1\}^n$. The adversary's task is to determine which the case is. Theorem 8 in [Rog04] proves that an adversary that makes at most $q$ such queries cannot distinguish these two cases with advantage[1] more than $9.5\frac{q^2}{2^n}+\varepsilon$, where $\varepsilon$ is the error term, $q$ is the number of encrypted 128-bit blocks, and $n$ is the block size of the underlying cipher in bits [CC07, Rog04]. However, this probability bound was improved by Minematsu to $4.5\frac{q^2}{2^n}+\varepsilon$ in his paper "Improved Security Analysis of XEX and LRW Modes" [Min07].

The real-world explanation for this security proof is based on the fact that no realistic adversary would have more information available than the adversary in the attack model described by Rogaway. This follows from the fact that the adversary in Rogaway's attack model is able to choose all the plaintext and ciphertext that is fed to the construction. Since the theorem (Theorem 8, [Rog04]) states that no adversary in that model can distinguish the construction from a collection of random permutations, it follows that no realistic adversary can distinguish between these cases with any significant advantage. This means that an attack would be just as successful against a collection of truly random permutations, one per each 128-bit block, as it would be against XEX.

Furthermore, the security proof in [Rog04] implies that the only attacks that are possible against XEX are the ones that are inherent from the use of transparent encryption with a narrow-block granularity. However, when the number of blocks $q$ that are available to the adversary approaches the birthday bound, there is a non-negligible probability that one is able to distinguish ciphertext produced by XEX from a random sequence of permutations [CC07].

### 4.1.2 XTS-AES Transform

As XTS-AES is an instantiation of the XEX scheme (with AES as the underlying block cipher), the authors of XTS-AES assume that the security proof for XEX holds for XTS [CC07], and can thereby also inherent the same probability bounds for an attacker to success. In Appendix D of IEEE Std 1619 the authors of XTS-AES argue that the inherit security from XEX yields a strong security guarantee as long as the same key is not used to encrypt much more than $2^{40}$ blocks (which corresponds to about 16 terabyte of data).

| | **Bytes** | **Queries ($q$)** |
|---|---|---|
| 1 Gigabyte | $1024^3$ | $2^{26}$ |
| 1 Terabyte | $1024^4$ | $2^{36}$ |
| 1 Petabyte | $1024^5$ | $2^{46}$ |
| 1 Exabyte | $1024^6$ | $2^{56}$ |

**Table 4.1:** *Table showing different amount of data encrypted with the same key and the number of queries (i.e. number of 128-bit blocks) that this data corresponds to.*

[1] In cryptography, the advantage is defined as the difference between the adversary's probability of being successful in an attack on the system and the probability of success by simply guessing. This can be expressed as *Adv(A) = Pr(Successful attack) – $\varepsilon$*, where $A$ is the adversary and $\varepsilon$ is the probability of success by guessing (also called the error term).

If we consider the case of one terabyte (which corresponds to $q = 2^{36}$), we can see from Table 4.2 that no attack can succeed with probability better than at most approximately $2^{-52}$ (i.e. approximately two in nine quadrillion).

The security guarantee deteriorates as more data is encrypted with the same key, as shown in Table 4.2. For example, when using the same key for a petabyte of data, attacks based on the birthday paradox have a success probability of at most approximately $2^{-32}$ (i.e. approximately one in four billion), and with one exabyte of data, the success probability is at most approximately $2^{-12}$ (i.e. approximate one in four thousand).

| **Queries ($q$)** | $9.5\frac{q^2}{2^{128}}$ | $4.5\frac{q^2}{2^{128}}$ |
|---|---|---|
| $2^{26}$ | $2^{-72.7521}$ | $2^{-73.8301}$ |
| $2^{36}$ | $2^{-52.7521}$ | $2^{-53.8301}$ |
| $2^{46}$ | $2^{-32.7521}$ | $2^{-33.8301}$ |
| $2^{56}$ | $2^{-12.7521}$ | $2^{-13.8301}$ |

**Table 4.2:** *Probability for an adversary to be successful in an attack based on the birthday paradox using the probability bounds deduced by Rogaway [Rog04] and Minematsu [Min07] for the XEX construction.*

But, note that the consequence of success is being able to distinguish XTS ciphertext from pseudo random permutations, not to recover the encryption key or plaintext. Figure 4.1 show a plot of the probability bound functions, which depict that to reach even a 1 % success probability in being able to distinguish XTS ciphertext from pseudo random permutations, a minimum of $2^{61}$ blocks, i.e. 32 exabyte, of data would have to be encrypted with the same key.

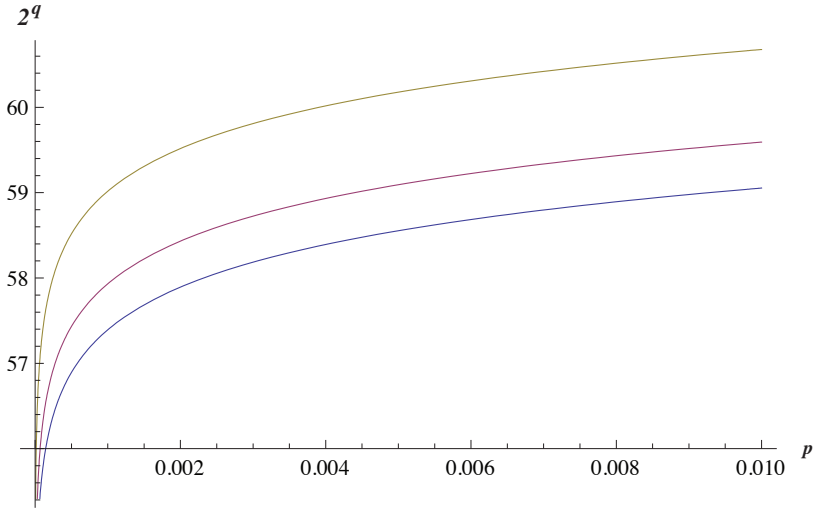



**Figure 4.1:** *Logarithmic plot ($2^q$) showing the probability ($p$) for an adversary to be successful in an attack based on the birthday paradox using the probability bounds $q^2/2^n$ (topmost graph), $4.5q^2/2^n$ (midmost graph), and $9.5q^2/2^n$ (bottommost graph).*

## 4.2 NIST Submission

As mentioned in section 3.1, IEEE submitted XTS to NIST for consideration as an approved encryption mode of operation for the AES block cipher, under NIST FIPS 140. As part of this process NIST sent out a request for public comments on XTS [32]. The main reason for this request was to give the cryptographic community (i.e. also those who did not participate in the development of XTS-AES) a chance to comment and evaluate the XTS-AES algorithm.

The following paragraphs aims to review and summarize the feedback provided by the cryptographic community. We have sought to include and describe all the well-documented responses submitted to NIST that in our opinion is interesting with respect to the security and viability of XTS. The complete set of comments submitted to NIST regarding XTS-AES can be found at [31, 33].

### 4.2.1 Security Related Feedback

#### Use of two keys instead of one

Liskov et al. criticizes the use of two keys in the XTS-AES algorithm in their response [LM08] to the request for comments on XTS-AES. They argue that two keys is unnecessary since the XEX construction is already been proven to be secure with only one key. Furthermore, the obvious advantage of using a single key is that the key length required would be halved, yet the level of security will remain effectively the same [LM08]. Moreover, they also point out that the standard does little to justify the choice to modify XEX so that two keys are used instead of one.

#### Security proof of XEX is not valid for XTS without modification

A second point of criticism against XTS-AES set forward by Liskov et al. is that the current security analysis of XTS-AES, included in the standard (and summarized in section 4.1) does not necessarily apply to XTS, since it does not adjust the analysis of XEX for the use of two keys. However, Liskov et al. provides such a security analysis in [LM08], that achieves a better security bound ($\frac{q^2}{2^n} + \varepsilon$), and avoids trying to apply the analysis of XEX improperly. Table 4.3 show the probabilities of being able to distinguish XTS ciphertext from pseudo random permutations when using the security bound deduced by Liskov et al.

| **Queries ($q$)** | $\frac{q^2}{2^{128}}$ |
|---|---|
| $2^{26}$ | $2^{-76}$ |
| $2^{36}$ | $2^{-56}$ |
| $2^{46}$ | $2^{-36}$ |
| $2^{56}$ | $2^{-16}$ |

**Table 4.3:** *Probability for an adversary to be successful in an attack based on the birthday paradox using the probability bounds deduced by Liskov et al. [LM08] for XTS-AES.*

Currently, all of the comments and responses (editorial and others) provided by Liskov et al. in [LM08] is listed under errata for IEEE Std 1619 at the official SISWG homepage [15], suggesting their feedback has been accepted.

### Ciphertext stealing

Michael Willett of Seagate points out in his response to the request for comment that the CTS implementation in XTS-AES makes the algorithm unnecessarily cumbersome [42]. He claims that the result of swapping the last two blocks is the loss of ability to access the data sequentially backwards and forwards, which in turn may also cause incompatibility issues with other mechanisms. He furthermore point out the absence of any security proof in the XTS-AES specification (or reference to such a proof) that the CTS implementation of XTS-AES will not worsen the security. Moreover, he suggests the use of a more recent definition of CTS [vT06] that do not include the swapping of the two last blocks.

Philip Rogaway also share the concern about the missing security proof for the CTS implementation: *"When the data unit (sector) is a multiple of 128 bits, each 128-bit block within the data unit should be separately enciphered as though by independent, uniformly random permutations. That part is clear. But what security property does one except for partial blocks? One might hope, for example, that the final 128+b bits (b<128) would likewise be enciphered as if by a strong PRP*[2] *"* [33].

In Liskov et al.'s response to the call for comment on XTS, they sketch a proof that XTS-AES's CTS implementation is sound if the basic scheme is sound. Furthermore, Matthew Ball, the Chair of IEEE SISWG, responds to these concerns by referring to the facts that NIST has proposed to incorporate a similar CTS mode for CBC [29] and that despite the lacking formal security proof CTS still has general approval in the cryptographic community [25].

### Inherent narrow-block weaknesses

Vijay Bharadwaj and Neils Ferguson of Microsoft submitted concern about the inherit narrow-block weaknesses of XTS-AES that allows fine-grained ciphertext manipulation attacks like data modification attack ([33], page 7).

In a data modification attack the adversary randomizes a block of data in such a way that introduce a security hole in the system whilst keeping the system functional. For example, modern operating systems have several settings that are important for the security of the system. Each of these settings is obviously an attack target. In a data modification attack, an adversary will try to find such a block of ciphertext that, when randomized, has a reasonable chance of changing a settings to something insecure; corrupting a access control function such that the system omits certain checks; or perhaps corrupting an input validation routine such that it does not clean up untrusted input. The narrow-block nature of the XTS-AES algorithm (i.e. that it works on 16-byte blocks of data at the time) makes this attack easy according to Bharadwaj and Ferguson. They suggest that a wide-block encryption method would not be as susceptible

[2] Pseudo Random Permutation

to such an attack since there are fewer data blocks to randomize. Additionally, when randomizing a wide-block, more of the overall data structure will be destroying yielding an increased chance of also damaging necessary parts of the data structure.

The P1619 Task Group acknowledges the narrow-block weaknesses in Appendix D of IEEE Std 1619 by confirming that an adversary has greater malleability with a 16-byte encryption mode like XTS, as opposed to wide-block algorithms like EME and XCB. The authors of XTS-AES justify the acceptance of the inherit narrow-block weaknesses by referring to the fact that narrow-block encryption is best suited only when a low-resource consuming cipher is needed (in terms of buffer, computation power, and latency) [25]. Thus, it is a tradeoff between efficiency and security.

#### Attack model addressed by XTS-AES

Another important comment provided by Michael Willett of Seagate, Niels Ferguson of Microsoft, and Philip Rogaway that the XTS-AES specification document, IEEE Std 1619, does not clearly specify what kind of attack model XTS-AES is suppose to address [42, 33]. Without a clearly formulated statement of what the application areas and security goals is, it is hard to analyze the proposal, or to determine how widely applicable XTS may be.

Matthew Ball recognizes that they could have added more details about all the applications that are suitable for XTS, but since this was considered a somewhat subjective area, it was omitted and ultimately left up to the designers and implementers to determine whether the XTS mode is appropriate for their specific design requirements [25].

### 4.2.2 Other Issues

#### XTS-AES specification only to be available through purchase

One of the topics that NIST wanted to be commented was the proposal for the approved specification to be available only by purchase from IEEE [32]. This proposal received very much criticism and was mostly commented as simply unacceptable by most of the responders to the call for comments on XTS-AES. The only proposal for an alternative solution was given by David Clunie ([33], page 3) who suggests that IEEE can be remunerated by the federal government in ways that do not involve purchase fees for the standard.

Matthew Ball explains their stand of why the standard must be associated with a publication fee in [25]. He argues that in the case of XTS-AES, the members of the P1619 Task Group (consisting of everyone from academic researchers to representatives of private companies) did not pay any fees to participate in the development, nor did NIST pay anything to develop this standard, and that this structure allowed the participants of top cryptographs to develop the details of the XTS-AES mode, free of cost to them. Naturally, when neither NIST is paying nor the P1619 Task Group members are paying, the publication cost is passed on to the user. He further argue that this is generally not an

issue for corporations, as they are expected to pay for standards as part of product development, in addition, he claims that many companies already have a company-wide access to the IEEE library, allowing them to access and download IEEE 1619 for no additional cost.

#### Concerns of intellectual property rights

There has also been some unease regarding the intellectual property (IP) rights of the XTS-AES algorithm. Matthew Ball wrote in his email to NIST dated September 1, 2008 that: *"Rogaway (the inventor of XEX, the basis of XTS) has no IP claims on XEX, nor knows of anyone else who does (see this e-mail [40]). NeoScale (now nCipher – soon to be acquired by Thales) submitted a blanket statement [44] of owning undisclosed IP for all of IEEE P1619, but there is no reason to believe that this IP covers XTS specifically."* [26].

From searches for patent applications owned by NeoScale performed by the P1619 Task Group, there was found some patents for general storage encryption methods and key management solutions as of January 2009 (complete list in [25]), but none that specifically covers XTS or XEX mode of encryption. The P1619 Task Group therefore dismisses the patent claims submitted by NeoScale as an ambiguity that exists for all technology [25]. At the time of writing, no new information or ruling regarding this matter has been made publicly available.

### 4.2.3 Changes

In light of the feedbacks received, the P1619 Task Group have suggested the following changes in order to facilitate a broad acceptance for XTS-AES as an NIST approved mode of operation [26]. Which of these that may or may not be included in the NIST Special Publication on XTS is not yet decided.

#### Allow using one key instead of two

The first change that the P1619 Task Group suggests NIST can make without effecting the security of XTS-AES is to allow for the use of one AES key instead of two [26]. This change is substantiated by the previously mentioned fact that the original XEX construction uses one key and at the same time is provable secure. This will effectively reduce the complexity of the ciphering processes, as key generation is a heavy duty operation.

#### Allow alternative ciphertext stealing method

Another change that the P1619 Task Group brings to NIST is to allow switching the order of the blocks created through CTS [25]. They argue that there is no security-related reason for the CTS implementation to be exactly as specified in IEEE Std 1619 [26].

#### Allow 192-bit AES keys

Although IEEE Std 1619 only specifies XTS-AES-128 and XTS-AES-256, i.e. XTS-AES encryption with 128-bit and 256-bit keys, the P1619 Task Group do not see any security-related reason for not also allowing 192-bit keys [26].

## 4.3 Attacks

The rest of this chapter will briefly go through some attacks that may be applicable to the narrow-block cipher mode XTS-AES. We have grouped the different attacks based on their area of application when regarding the following two attack scenarios:

1. The case of a *stolen storage device* already employing full disk encryption with the possibility that the adversary (at best) has partial knowledge to the contents of the stolen disk (i.e. the attacker is able to initiate ciphertext-only attacks and known-plaintext attacks).

2. The case where the adversary is able to monitor and *manipulate the ciphering process* of a full disk encryption solution (i.e. the attacker is able to initiate chosen-plaintext attacks and chosen-ciphertext attacks).

### 4.3.1 Scenario: Stolen Storage Device

In this scenario the adversary has stolen the encrypted storage device. Attacks applicable in this scenario assume that the adversary is computationally unbounded in terms of time and storage resources.

#### Exhaustive key search attack

Exhaustive key search attack refers to the attempt of systemically guessing a cryptographic key. This is done by trying every possible key to decipher a ciphertext until one key result in some intelligible plaintext. The complexity of this attack can be described by the number of tried decryptions, which is, on average $2^{k-1}$ for a block cipher with a key size $k$ [JBSK01]. An exhaustive key search can be mounted on any cipher and sometimes a weakness in the key schedule of the cipher can help improve the efficiency of such an attack.

An exhaustive key search is a typical brute-force attack that can be applied not only to block ciphers, but to any cryptographic system that's protection relies on the secrecy of the key [JBSK01]. Hence, it is important that the systems key size is large enough to make the exhaustive key-search attack infeasible due to time complexity.

The feasibility of an adversary being able to launch an exhaustive key search attack against a stolen disk that employs XTS-AES encryption is dependent on AES's ability to withstand a such attack, and since AES is unlikely to be broken by an exhaustive key search attack in the foreseeable future, so is XTS-AES. But, the condition is of course that the master key (used by the KeyExpansion stage of AES to derive the rounds keys) is not trivially predictable.

#### Birthday attack

The idea behind this attack originates from the birthday paradox[3] that states that a given a group of 23 randomly chosen people, the probability of at least

[3]Its actually not a paradox in the sense of leading to a logical contradiction, but is called a paradox because the mathematical truth contradicts naïve intuition.

two of them having the same birthday is more than 50 % [Fel68]. The mathematics behind the birthday paradox can be translated into the well-known cryptographic attack called the birthday attack.

The birthday attack is independent of the underlying algorithm used because it relies on the ability to find two ciphertexts, such that $f(c_1) = f(c_2)$, for a given function $f$ where $c_1 \neq c_2$. The method used to find a collision is to evaluate the function $f$ for different ciphertexts that may be chosen randomly or pseudo randomly until the same result is found more than once. This attack is limited by the space complexity, as acquiring enough ciphertexts to successfully launch this attack may be hard. On average one needs $2^{\frac{n}{2}}$ ciphertexts to successfully perform a birthday attack, where $n$ is the block length of the cipher [Tan02].

This attack is applicable to disk encryption as well, especially when there is no tweak or chaining between blocks used to further variate the output of each block cipher encryption. Nevertheless, considering the scenario of a stolen storage device encrypted with a narrow-block cipher, this yields a whole disk of 128-bit blocks of ciphertext; slightly relaxing the space complexity of a birthday attack.

#### Dictionary attack

Encryption algorithms usually use a password to generate the key or keys used in ciphering operations. This exposes inherent weaknesses in the security, as the complexity is entirely dependent on the password strength, or "unguessability". An attack that exploits this weakness by systematically testing passwords likely to succeed is called a dictionary attack.

Dictionary attacks are somewhat similar to exhaustive key search attacks, but instead of trying every possible key (which usually is at least 128-bit), a dictionary attack tries every possible password likely to succeed, typically derived from a list of words in a dictionary. Generally, dictionary attacks succeed because many people have a tendency to choose passwords which are short (usually 7 characters, plus or minus two [Mil56]), single words found in dictionaries or simple easily-predicted variations on words, such as appending a digit.

While an ordinary dictionary attack is limited by time complexity (i.e. the time it takes to test all possible passwords), it is possible to achieve a time-space tradeoff through pre-computation by encrypting and storing a list of encrypted dictionary words, sorted by the encrypted value. This requires a considerable amount of preparation time, but makes the actual attack almost instantaneous. The time-space tradeoff means that the attack becomes limited by space complexity (i.e. storing all possible encrypted dictionary words) instead of by time complexity.

All cryptosystems, including disk encryption solutions that use passwords to derive the cryptographic key(s) later used in ciphering operations are susceptible to this attack. Unless, there exists a policy that limits the number of authentication attempts that can be made.

#### Cold boot attack

Cold boot attack is a type of password attack susceptible to most software-based encryption schemes [6]. The fact that software-based encryption solutions store the encryption key(s) in the computer memory, in order to be able to do on-the-fly ciphering operations, is exploited in this attack. These keys, if extracted, cause a collapse in the software-based security scheme.

This attack relies on the data remanence property of computer memory, whereby data bits can take up to several minutes to degrade after power has been removed. This time can be further prolonged by cooling down the memory, with say compressed air or liquid nitrogen, in which case it will take up to several hours for data bits to decay [HSH+08]. The actual attack is usually carried out by "cold-booting" a machine already running an operating system and a software-based disk encryption scheme, then dumping the contents of memory before the data disappears.

Although the basic issue has been known for several years [6], this attack was formally proposed by Halderman et al. in their paper "Lest We Remember: Cold Boot Attacks on Encryption Keys" [HSH+08]. In this paper, they demonstrate that it is possible to leverage the remanence effects in memory modules by performing a "cold-boot" the target computer, load a custom OS extracting the memory to an external drive, locate the key material and decrypt hard drives automatically. To compensate for the risk of bit errors that has already decayed in memory (during the attack), they also suggest methods for correcting such errors by utilizing the inherited "dead state" structure of memory modules and an error-correcting code [HSH+08].

This attack is only applicable if the adversary steals a computer that is already running or in sleep/suspend mode – in which cases the memory is still active [6]. Since this attack is independent of the underlying cipher, it is also applicable to software-based encryption solutions using XTS-AES.

### 4.3.2 Scenario: Manipulate Disk Encryption Activity

In this scenario the adversary is able to monitor and manipulate data being read and written to and from the storage device (i.e. the adversary is able to launch both chosen-plaintext attacks and chosen-plaintext attacks). A real-world example of this can be the case of out-sourced storage, where the adversary is in complete control of the storage medium either by network or physical access. Attacks applicable in this scenario assume that the adversary is computationally bounded in terms of time and storage resources.

#### Non-malleability

Non-malleability is a property for cryptographic algorithms formally defined by Dolev et al. [DDN00]. An encryption algorithm is considered malleable if it is possible to an adversary to transform a ciphertext into another ciphertext which decrypts to a related plaintext. A overly simplified definition is, given an encryption of plaintext $m_1$, $E_k(m_1)$, it is possible to generate another ciphertext

$E_k(m_2)$ which decrypts to $m_2$ that is related to $m_1$, for a known $E_k$, without necessarily knowing or learning $m_1$.

Malleability is often undesired[4] since it allows an adversary to modify the contents of a confidential message. If an adversary is able to guess the syntax or format of the message and the applied encryption scheme is malleable, this can be exploited. Consider the following example where encryption is performed with a stream cipher, which is inherently malleable. Suppose a bank encrypts its financial information, and a customer sends an encrypted message containing, say, "`TRANSFER $0000100.0 TO ACCOUNT #1985`". If an attacker knows or is able to guess the format of the encrypted message and modify the message in transit, the attacker could be able to change the amount of the transaction, or recipient of the funds, e.g. "`TRANSFER $0100000.0 TO ACCOUNT #1337`".

Stream ciphers are bit-level malleable[5] as it is trivial to predict the change in plaintext if one is to modify the ciphertext. Block ciphers on the other hand, are usually harder to manipulate in the same way, and malleability is often only achieved at a block-level. However, cipher modes like CBC have constructional vulnerabilities making them bit-level malleable. When considering the malleability of XTS-AES, one must take into account that all modes with a single AES pass are exposed to malleability of encrypted text [4]. Thus, we must also consider XTS-AES as malleable. XTS is however "less malleable" than other narrow-block ciphers mode like CBC-AES. For XTS an adversary cannot randomize with a granularity of less than 16-bytes, whereas with CBC, the adversary can modify specific bits within a 16-byte block – with the cost of randomizing the previous 16-bytes [25].

### Cut-and-paste attack

A cut-and-paste attack is an attack in which the adversary substitutes a section of ciphertext with a different section that looks like (but is not the same as) the one removed. The substituted section appears to decrypt normally, along with the authentic sections, but results in a plaintext that serves a particular purpose for the adversary. Consider the following "mix and match" weakness, if $E_k(P_0P_1...P_m) = C_0C_1...C_m$ and $E_k(\bar{P}_0\bar{P}_1...\bar{P}_m) = \bar{C}_0\bar{C}_1...\bar{C}_m$ then $E_k(P_0\bar{P}_1...P_m) = C_0\bar{C}_1...C_m$. Essentially, the adversary cuts one or more sections from the ciphertext and reassembles these sections so that the decrypted data will result in coherent but invalid information. Cut-and-paste attack is a type of message modification attack: the adversary removes a message from a message flow (e.g., network traffic), alters it, and reinserts it.

XTS-AES is susceptible to this attack mainly since the encryption process does not produce any authentication tags to ensure data integrity. This means that any ciphertext (original or modified by the adversary) will decrypt as some plaintext and there is no built-in mechanism to detect possible alterations. The

[4] Some cryptosystems are malleable by design [DDN00]. Such schemes are known as homomorphic encryption schemes.

[5] If a cipher is said to be bit-level malleable, it means that its susceptible for a bit-flipping attack, i.e. the adversary is able to change one bit in the ciphertext that will yield a predictable change of one bit in the plaintext.

countermeasure used by the XTS-AES algorithm to mitigate cut-and-paste attacks is by using the position address of the data as an additional input in the encryption and decryption processes. Incorporating this position information makes it possible to cryptographically hide the fact that the same data block encrypted with the same key will yield the same ciphertext, and also prevents any "mix and match" between different positions. But, in the scenario where the adversary is able to monitor and possibly modify the ciphering process, it is also reasonable to assume that he or she can deduce the position address.

## 4.4 Summary

In this chapter, we assessed the security of XTS-AES. By doing so we have made several observations, of which a summary follows.

- By reviewing the security proof of XEX/XTS, we found that to even reach a 1 % probability of successfully being able to distinguish XTS ciphertext from pseudo random permutations, a minimum of 32 exabyte of data have to be encrypted with the same key. When we recognize that this is the complexity of being able to initiate an attack relying on ciphertext similarities (i.e. birthday attack), we may rightfully state that the computational security of XTS-AES is indeed strong[6].

- We further noted that wide-block encryption modes are generally considered to provide better security than narrow-block modes [CC07, 25]. (The wide-block initiative P1619.2 will thus be very interesting to follow). The particular strength of the XTS-AES cipher mode is storage encryption systems where the applications using the plaintext is able to detect when 16-byte blocks are randomized, and where using wide-block encryption or authenticated encryption is not feasible due to limited processing power, latency requirements, or space limitations.

- When reviewing potential attacks against XTS-AES, we found that the common for all of these is that they are either generic attacks (applicable to all cryptosystems) or that they rely on the inherent nature of XTS-AES[7]. Namely, that XTS-AES is a transparent narrow-block cipher, which in turn makes it vulnerable to attacks that rely on the absence of any data integrity check prior to decryption. Attacks which exploit the latter quality include data modification attacks like cut-and-paste attack and malleability attack. The other attack types, namely exhaustive key search attack, dictionary attack, and cold boot attack are all generic attacks. This means that they do not attack on the cipher mode it self, but rather the keys and passwords used for encryption. Hence, the only countermeasure to such attacks is to implement security mechanisms and policy's independent of the encryption mode.

[6] We emphasize that although there is mismatch between different papers discussing the security of XTS-AES, the computational security of XTS-AES is strong no matter which security bound (i.e. $9.5q^2/2^n$, $4.5q^2/2^n$, or $q^2/2^n$) one assume applies. The example above, holds for all the discussed bounds.

[7] With the exception of cold boot attack, which relies on the the data remanence property of computer memory [HSH+08].

# Chapter 5

# Performance Benchmark

*No amount of experimentation can ever prove me right; a single experiment can prove me wrong.*

**Albert Einstein**

We believe that the best way to assess the performance of XTS-AES based disk encryption, is to provide real-world measurements on how it affects a computer's hard disk performance.

This chapter will present the procedure, test bench, test cases, and testing methodology used to acquire objective performance measurements. The results and corresponding analysis is however presented in chapter 6. Note that this performance benchmark is limited to software-based encryption only, and does not consider hardware-based encryption.

## 5.1 Procedure

The performance assessment was conducted by benchmarking three different computers hard disk performances when (1) no disk encryption was present and (2) when the disk employed transparent XTS-AES-256 full system disk encryption. We repeated the benchmarking in case (1) and (2) for Windows XP, Windows Vista, and Windows 7 (Beta) on each computer. The performance attributes of interest was; *write speed*, *read speed*, and *CPU usage* during disk operations. We chose to focus on the latter attributes because we believe these are of relevance when considering the performance of a hard disk.

When we later present and discuss the results, we will focus on the *change* in performance when full system disk encryption is introduced. In the case of transfer speeds (i.e. write and read speed) we are interested in the *percent wise change*, since this is a metric that makes it possible to compare different computers' performance with each other. But, in the case of CPU usage during disk operations, this is an attribute that is already measured in percent, and thus it is of most interest to compare the *numerical change* that occurs when introducing XTS-AES-256 full system disk encryption.

## 5.2 Test Bench

Before we proceed with the tests and test cases, this section will present the test bench, consisting of computer hardware and software used in the performance benchmark.

### 5.2.1 Computers

The computer hardware was chosen with respect to the following priority: diversity, relevance, and availability (in that order). Specifications of the computers used in the performance benchmarking of XTS-AES follows below.

#### Computer 1

Manufacturer: *Hewlett-Packard*
Model: *Pavilion zd8000*
Type: *Laptop*
Chipset: *Intel 915 Express Chipset Family*
CPU type: *Intel Pentium 4 650 processor w/Hyper Thread Technology*
CPU clock rate: *3.40 GHz, 2 MB L2 cache, 800 MHz FSB*
Number of physical CPU cores: *1*
Memory: *1 GB 533 MHz DDR2 SDRAM*
Hard disk: *Seagate 100GB Momentus 5,400 rpm (ST9100822A)*

#### Computer 2

Manufacturer: *Fujitsu Siemens*
Model: *Scaleo T-360*
Type: *Desktop*
Chipset: *Intel 915 Express Chipset Family*
CPU type: *Intel Pentium 4 560 processor w/Hyper Thread Technology*
CPU clock rate: *3.60 GHz, 1 MB L2 cache, 800 MHz FSB*
Number of physical CPU cores: *1*
Memory: *3 GB 400 MHz DDR2 SDRAM*
Hard disk: *Western Digital 160 GB 7,200 rpm (WD1600JD-55HBB0)*

#### Computer 3

Manufacturer: *Hewlett-Packard*
Model: *Compaq dc7900*
Type: *Desktop*
Chipset: *Intel Q45 Express Chipset Family*
CPU type: *Intel Core 2 Duo E8400 processor*
CPU clock rate: *3.00 GHz, 6 MB L2 cache, 1333 MHz FSB*
Number of physical CPU cores: *2*
Memory: *4 GB 800 MHz DDR2 SDRAM*
Hard disk: *Western Digital 160 GB 7,200 rpm (WD1600AAJS-60B4A)*

Before use and between testing with different operating systems, each computer was formatted and fresh copies of software and appropriate system drivers were installed.

### 5.2.2 Software

Before deciding which software to use in our performance benchmark, we did an extensive survey of available disk encryption applications and benchmarking tools. The following paragraphs state the motivation for using the chosen software, namely *TrueCrypt* for disk encryption and *HD Tune Pro* for disk benchmarking.

#### TrueCrypt 6.1a

TrueCrypt [45] was presented in section 2.4.1. We used TrueCrypt to apply *full system disk encryption* using XTS-AES-256 (i.e. XTS mode encryption with AES using 256-bit keys). TrueCrypt was chosen because of several reasons; free and open source software, has good reputation in the security society, is well-documented, continuously updated, and last but not least because it is the only encryption application for the windows-platform that supports XTS-AES and full system disk encryption.

Step-by-step instructions on how full system disk encryption were applied using TrueCrypt 6.1a is given in Appendix A.

#### HD Tune Pro 3.5

HD Tune Pro [43] is a multi-purpose hard disk utility for the windows-platform. HD Tune Pro features disk health checks, error scanning, temperature monitoring, benchmarking of transfer rates (read and write), and measurements of access time, burst rate, and CPU usage during disk operations.

After a comprehensive trail and error process pursuing the most suitable benchmark application for our scope, we found HD Tune Pro to be the best software for measuring the performance attributes of interest, namely *write speed*, *read speed*, and *CPU usage*.

#### Operating systems

As mentioned in beginning of this chapter, we repeated the benchmarking on the following windows operating systems:

- *Windows XP Professional* with Service Pack 3
- *Windows Vista Business Edition* with Service Pack 1
- *Windows 7 Ultimate Edition (Beta Build 7057)* as the bleeding edge Microsoft operating system was also included in the testing.

It is worth mentioning that the Ultimate and Enterprise Editions of Windows Vista and Windows 7 already includes a built-in full system disk encryption option, called BitLocker [23]. The reason for not using this to perform full system disk encryption was that BitLocker only features encryption based on AES in CBC mode at the time of writing [Fer06].

## 5.3 Tests

### 5.3.1 File Benchmark

The first test of interest was a file benchmark performed to measure the read and write speed of the hard disk.

| | |
|---|---|
| **Test-ID:** | `File_Benchmark` |
| **Purpose:** | Measure read speed and write speed with the hard disk utility *HD Tune Pro 3.5*. |
| **Prerequisites:** | No applications running in the background. No wired or wireless network connections enabled. |
| **Steps:** | **1.** Open HD Tune Pro 3.5<br>**2.** Choose the *File benchmark* pane and press *Start*<br>**3.** After the file benchmark is complete, copy the complete result set to a log file<br>**4.** Repeat the steps **2**–**3** *ten* consecutive times<br>**5.** Close HD Tune Pro 3.5 |
| **Notes:** | The benchmark performed in this test returns write and read speeds for different block sizes. The block size of interest during this test is 64 KB, and thus the only result which is presented later on. This means that the speeds measured will tell us how fast data is written or read when the file operations is done on data blocks of 64 KB in size. The HD Tune manual recommends that the data block size is set to 64 KB, as lower values may give lower test results [43]. |

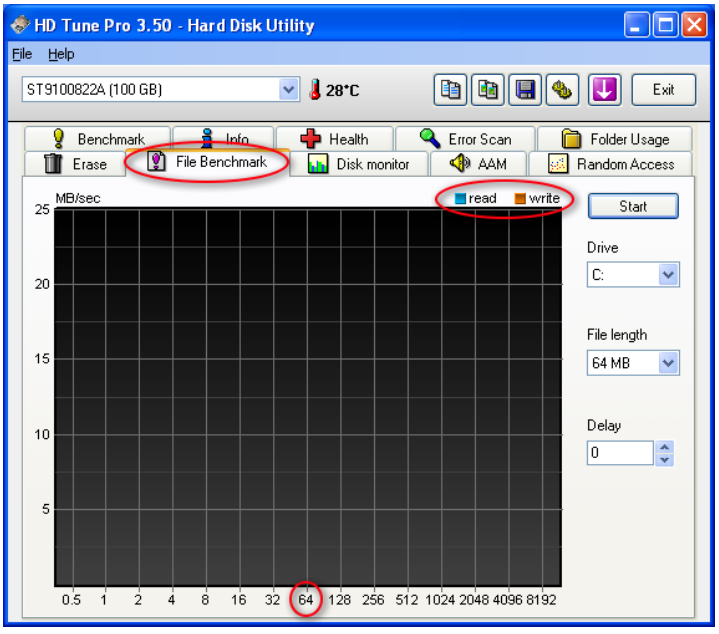



**Figure 5.1:** *File benchmark performed with HD Tune Pro 3.5*

### 5.3.2 CPU Benchmark

The second, but nonetheless important test was to measure the CPU usage during disk operations.

| | |
|---|---|
| **Test-ID:** | `CPU_Benchmark` |
| **Purpose:** | Measure the CPU usage during disk operations with the hard disk utility *HD Tune Pro 3.5*. |
| **Prerequisites:** | No applications running in the background. No wired or wireless network connections enabled. |
| **Steps:** | **1.** Open HD Tune Pro 3.5<br>**2.** Choose the *Benchmark* pane and press *Start*<br>**3.** After the file benchmark is complete, copy the complete result set to a log file<br>**4.** Repeat the steps **2**–**3** *ten* consecutive times<br>**5.** Close HD Tune Pro 3.5 |
| **Notes:** | The benchmark performed in this test returns transfer rate (read only), access time, burst rate and CPU usage. Of these, it is only the CPU usage that is of interest, and thus the only result which is presented later on. The CPU usage tells us how much CPU time (in %) the system is using to *read* data from the disk. Although it would be interesting to also measure how much CPU time the system is using to *write* data, this was not possible with the current settings. |

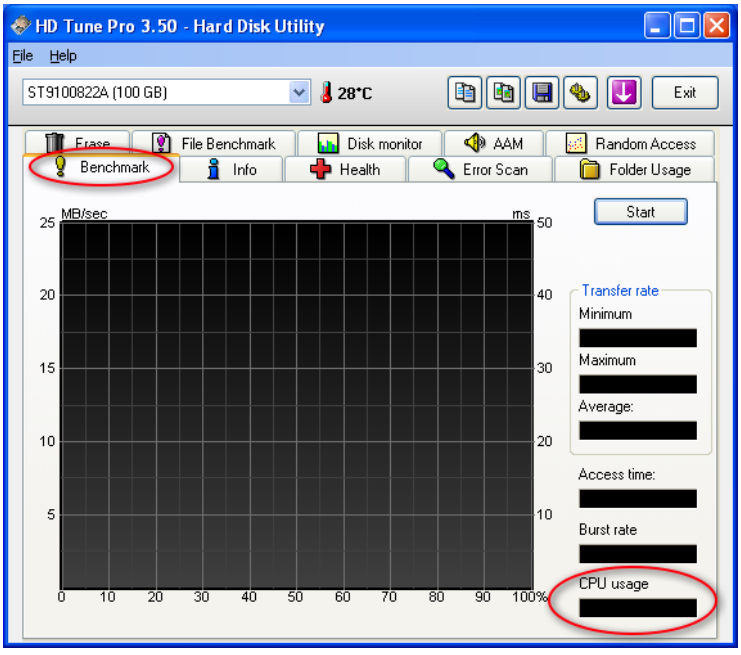



**Figure 5.2:** *CPU benchmark performed with HD Tune Pro 3.5*

## 5.4 Test Cases

The following two tables describes the test cases used to compare the performance variables obtained when (1) no disk encryption was present and (2) when the disk employed transparent XTS-AES-256 full system disk encryption.

When reading through the test cases, it might be useful and clarifying to go back and skim through `File_Benchmark` (section 5.3.1) and `CPU_Benchmark` (section 5.3.2) when mentioned, as they are used as subroutines.

### 5.4.1 Without Disk Encryption

| **Testcase-ID:** | `No_Encryption` |
|---|---|
| **Purpose:** | Measure the write speed, read speed, and CPU usage when no disk encryption is applied. |
| **Prerequisites:** | Clean installation of operating system with no other software than HD Tune Pro 3.5 and necessary system drivers installed. |
| **Steps:** | **1.** Perform `File_Benchmark`<br>**2.** Perform `CPU_Benchmark` |
| **Notes:** | It is expected that when `No_Encryption` is performed on different operating systems, it will most likely give different results (even when using the same computer). This might sound odd, since it is the same hardware being tested. Although not desirable, the truth is that different operating systems might have different device drivers for the same piece of hardware, which in turn may affects the performance achieved. |

### 5.4.2 With Disk Encryption

| **Testcase-ID:** | `XTS-AES-256_Encryption` |
|---|---|
| **Purpose:** | Measure the write speed, read speed, and CPU usage when XTS-AES-256 full system disk encryption is applied. |
| **Prerequisites:** | Clean installation of operating system with no other software than TrueCrypt 6.1a, HD Tune Pro 3.5, and necessary system drivers installed. XTS-AES-256 full system disk encryption applied by TrueCrypt as described in Appendix A. |
| **Steps:** | **1.** Perform `File_Benchmark`<br>**2.** Perform `CPU_Benchmark` |
| **Notes:** | It is expected that when `XTS-AES-256_Encryption` is performed on different operating systems, it will most likely give different results (even when using the same computer). |

## 5.5 Testing and Benchmarking Methodology

This section will give an introduction to test methodology that should be properly addressed to ensure the quality of a benchmark like this. In addition, we define arithmetic average, standard deviation, standard error, and confidence interval.

### Quantitative versus Qualitative

Data can be described in quantitative terms or qualitative terms. The way we typically define them, we call data "quantitative" if it is in numerical form and "qualitative" if it is not. Quantitative research is identified by the fact that it involves analysis on data that exist in a range of magnitude (i.e. it is quantifiably measurable). Examples of physical quantities are distance, mass and time. Qualitative research on the other hand, involves analysis of data such as words (e.g., from interviews), pictures (e.g., video), or objects (e.g., an artifact).

Our performance benchmark might hence be considered a strictly quantitative research as our aim is to provide quantifiable performance measurements.

### Variable Control

Testing can be rendered completely useless if the number of variables involved are not reduced to an absolute minimum. For experiments, the ideal is to have only one variable. In a real world environment however, only having one variable is not possible, but the good experimenter attempts to reduce the effects of anything that might affect the test. When variables are not kept in check, accuracy may be reduced, although the perceived precision may be high.

In our testing, we tried to keep the number of variables at an minimum by facilitating that the least amount of background processes was running: a fresh boot was performed before every test case was run, wired and wireless network connections was disabled, and no other software other than necessary system drivers, TrueCrypt, and HD Tune Pro was installed or executed at any time.

### Significant Digits

The basic idea here is that you do not keep more digits than you measure. For example, when putting raw numbers into equations or averaging data, some calculators will return more digits than what is significant. The calculator may give a mean of 3.33333. If your only were to include three significant digits in your data (e.g., 3.33, 3.26, and 3.45), the correct number of significant digits are not kept. This can lead to the idea that the data is more or less precise than in actuality.

Our performance benchmark relates exclusively to disk speeds and CPU usage. To present these numbers, we will be using up to four significant digits for both speed (measured in MB/s) and CPU usage (CPU time in %). Furthermore, all speed measurements are presented in megabytes per second (MB/s), where one megabyte equals to $1,048,576$ bytes (i.e. $1,024$ kilobytes).

### Repeatability of Results

The repeatability of a test is the most telling gauge of how well an experiment has been done. If the tester and a third party follow the same guidelines and get the same results, the test results may be considered repeatable.

We have facilitated for the readers of this thesis to be able to repeat this performance benchmark by accurately listing the test procedure, test bench, and test cases used. Furthermore, the ZIP file attachment to this thesis includes the software applications used in the benchmark (see Appendix C).

### Sample Size

The more times a test is done under the correct conditions, the more likely spurious results (small mistakes in each individual test) will not adversely affect the final averages. Each test instance should be tested multiple times, and it is preferable to have more than one of the items to perform tests on, as some examples may perform better or worse than others.

In our testing, we had a sample size of ten. As previously described we used three different computers, each with different specifications. Each computers disk performance was tested under three operating systems. Figure 5.3 depicts a tree-structure showing the test order. After a clean installation of Windows XP, the test case `No_Encryption` was performed (consisting of `File_Benchmark` and `CPU_Benchmark`, which in turn each consists of ten repetitions – hence the sample size of ten). Then, the test case `XTS-AES-256_Encryption` was performed (consisting of the same tests as in the latter case, but now the hard disk employed transparent full system disk encryption). This process was then repeated for Windows Vista and Windows 7.

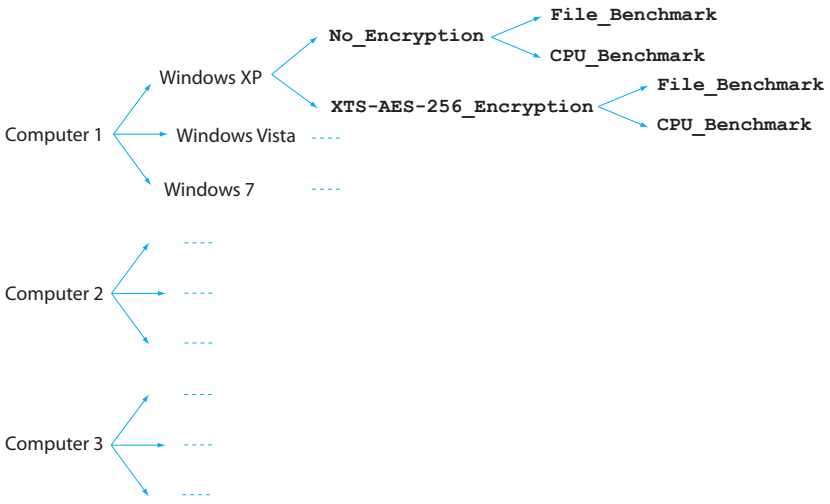



**Figure 5.3:** *Both write speed, read speed and CPU usage was measured ten times in each case, hence the sample size of ten. These benchmarks where repeated both without and with disk encryption, with three different operating systems, on three different computers, yielding a total of 360 benchmark tests.*

### Arithmetic Average

The arithmetic average (or mean) of a list of numbers is the sum of the entire list divided by the number of items in the list (shown in Equation 5.1). If the list is a statistical sample, we call the resulting statistic a sample mean.

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.1}$$

After each test (i.e. `File_Benchmark` and `CPU_Benchmark`) was performed, the sample mean of the result set was calculated. For the sake of simplicity, it is the sample means that are presented in chapter 6, while the exhaustive result sets (i.e. all ten measurements in each case) are listed in Appendix B.

### Standard Deviation/Standard Error

The standard deviation (SD) of a population or sample set is a measure of the data's spread of values (expressed in Equation 5.2). Standard error (SE) is the standard deviation over the root of the number of samples (expressed in Equation 5.3). Thus, standard error takes into account the number or samples in the experiment.

$$SD = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.2}$$

$$SE = \frac{SD}{\sqrt{n}} \tag{5.3}$$

Simply put, a small standard deviation/standard error means that most of the values recorded during testing are within a small range of the arithmetic average of the data. In other words, the smaller the standard deviation/standard error, the more the numbers are to be trusted. The standard deviation and standard error of each result set is listed in Appendix B.

### Confidence interval

A confidence interval (CI) is an interval estimate of a population or sample parameter. Instead of estimating the parameter by a single value, an interval likely to include the parameter is given. Thus, a confidence interval will indicate the reliability of an estimate. How likely the interval is to contain the parameter is determined by the confidence level and a corresponding confidence coefficient (expressed as $\alpha$ and $z_\alpha$ in Equation 5.4). Increasing the desired confidence level will widen the confidence interval.

$$Pr(\bar{x} - SE \cdot z_\alpha < x < \bar{x} + SE \cdot z_\alpha) = 1 - \alpha \tag{5.4}$$

We chose to calculate a 95 % confidence interval ($\alpha = 0.05$) using a Student's t-distribution for each result set (corresponding to $z_\alpha = 2.228$ when considering a sample size of 10). This makes us able to claim that there is a 95 % probability that the "correct" value will be within in the confidence interval. The interval for each result set is listed in Appendix B. The reason for using Student's t-distribution is that the standard deviation is not given in beforehand, but rather has to be calculated after the fact.

# Chapter 6

# Results

*If we knew what we were doing, it wouldn't be research.*

**Albert Einstein**

*In theory, there is no difference between theory and practice. But, in practice, there is.*

**Jan L. A. van de Snepscheut**

This chapter will present the results from the performance benchmark described in the previous chapter. In addition we will also provide an analysis of the results by discussing causality, possible consequences, limitations and possible sources of error, along with suggestions for further work.

Please note that the abbreviated results are presented in this chapter, while the exhaustive result sets are listed in Appendix B.

## 6.1 Benchmarking Results

Before we proceed with the test results, we want to emphasize that we are interested in the *change* in performance when introducing XTS-AES-256 full system disk encryption. As we recall from section 5.1, we stated that in the case of transfer speeds we are interested in the *percent wise change*, while in the case of CPU usage during disk operations, we are interested in the *numerical change*.

To ensure the validity of our results, we make the following assumptions. We assume that the results obtained by using TrueCrypt 6.1a to employ XTS-AES-256 full system disk encryption is representative for the actual performance of a software-implementation of XTS-AES using 256-bit keys as specified by the IEEE Std 1619. We also assume that the result obtained by taking the average of each respective performance attribute under all three operating systems yield a representative metric of the write speed, read speed and CPU usage for each computer.

### 6.1.1 Write Speed

Table 6.1 and Figure 6.1 shows the average write speeds, both without and with disk encryption for each of the operating systems on the different computers. In addition, the numerical and percent wise change in each case is also listed.

| | **Write speed** | | **Difference** | |
|---|---|---|---|---|
| | *Without disk encryption* | *With disk encryption* | *Numerical* | *Percent* |
| **Computer 1** | | | | |
| *Windows XP* | 27.2 MB/s | 21.2 MB/s | −6.0 MB/s | **−22.1 %** |
| *Windows Vista* | 26.8 MB/s | 20.8 MB/s | −6.0 MB/s | **−22.4 %** |
| *Windows 7* | 27.2 MB/s | 20.6 MB/s | −6.6 MB/s | **−24.2 %** |
| **Computer 2** | | | | |
| *Windows XP* | 59.2 MB/s | 42.1 MB/s | −17.1 MB/s | **−28.8 %** |
| *Windows Vista* | 51.4 MB/s | 40.4 MB/s | −11.0 MB/s | **−21.5 %** |
| *Windows 7* | 60.6 MB/s | 41.2 MB/s | −19.4 MB/s | **−32.0 %** |
| **Computer 3** | | | | |
| *Windows XP* | 93.9 MB/s | 63.0 MB/s | −30.9 MB/s | **−32.9 %** |
| *Windows Vista* | 87.1 MB/s | 65.1 MB/s | −22.0 MB/s | **−25.3 %** |
| *Windows 7* | 98.6 MB/s | 54.4 MB/s | −44.2 MB/s | **−44.9 %** |

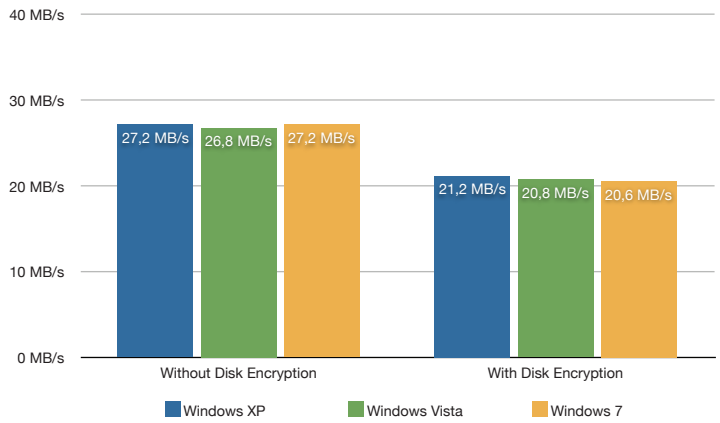
**Table 6.1:** *Average write speeds during benchmark*

We observe that,

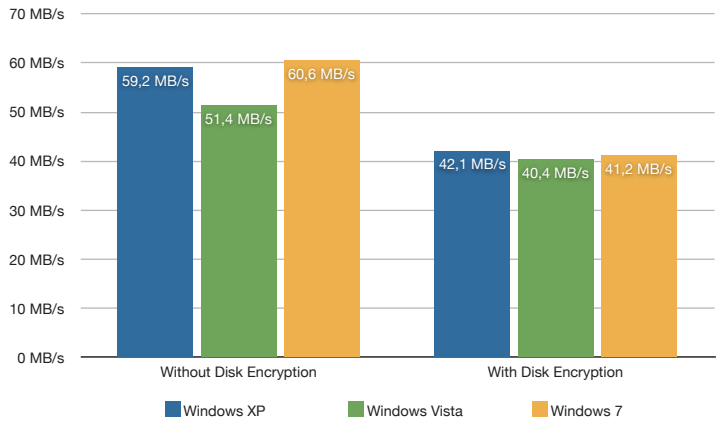
- Computer 1's write speed is approximately 27 MB/s when operating without disk encryption and approximately 21 MB/s when employing full system disk encryption, yielding a decrease of approximately *−22 %*.
- Computer 2's write speed is approximately 57 MB/s when operating without disk encryption and approximately 41 MB/s when employing full system disk encryption, yielding a decrease of approximately *−28 %*.
- Computer 3's write speed is approximately 93 MB/s when operating without disk encryption and approximately 61 MB/s when employing full system disk encryption, yielding a decrease of approximately *−34 %*.

The percent wise change in write speed is different for each computer. Although we expected some difference, we did not expect the gap to be of this magnitude; the lowest percent wise decrease in write speed is approximately −22 % and the highest percent wise decrease in write speed is approximately −34 %. However, we identify an interesting pattern; the higher the disks write speed is without disk encryption, the larger the percent wise decrease is measured when full system disk encryption is applied. But since correlation does not necessarily imply causation, we cannot say for sure.

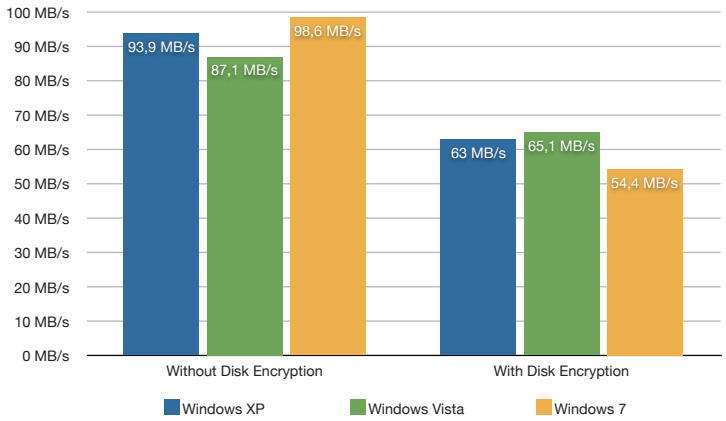
**Summary.** Our measurements show that the percent wise decrease in write speed is approximately −28 % (±6 %) when using full system disk encryption.




**(a)** *Average write speeds for computer 1 during benchmark*




**(b)** *Average write speeds for computer 2 during benchmark*




**(c)** *Average write speeds for computer 3 during benchmark*

**Figure 6.1:** *Average write speeds during benchmark for (a) computer 1, (b) computer 2, and (c) computer 3 using different operating systems.*

### 6.1.2 Read Speed

Table 6.2 and Figure 6.2 shows the average read speeds, both without and with disk encryption for each of the operating systems on the different computers. In addition, the numerical and percent wise change in each case is also listed.

| | **Read speed** | | **Difference** | |
|---|---|---|---|---|
| | *Without disk encryption* | *With disk encryption* | *Numerical* | *Percent* |
| **Computer 1** | | | | |
| *Windows XP* | 28.7 MB/s | 19.7 MB/s | −9.0 MB/s | **−31.2 %** |
| *Windows Vista* | 28.6 MB/s | 19.2 MB/s | −9.4 MB/s | **−32.8 %** |
| *Windows 7* | 28.5 MB/s | 18.8 MB/s | −9.7 MB/s | **−34.1 %** |
| **Computer 2** | | | | |
| *Windows XP* | 57.2 MB/s | 41.6 MB/s | −15.6 MB/s | **−27.2 %** |
| *Windows Vista* | 50.7 MB/s | 39.8 MB/s | −10.9 MB/s | **−21.5 %** |
| *Windows 7* | 57.5 MB/s | 40.9 MB/s | −16.6 MB/s | **−28.9 %** |
| **Computer 3** | | | | |
| *Windows XP* | 93.9 MB/s | 66.4 MB/s | −27.5 MB/s | **−29.3 %** |
| *Windows Vista* | 88.4 MB/s | 69.6 MB/s | −18.8 MB/s | **−21.3 %** |
| *Windows 7* | 98.6 MB/s | 62.8 MB/s | −35.8 MB/s | **−36.3 %** |

**Table 6.2:** *Average read speeds during benchmark*

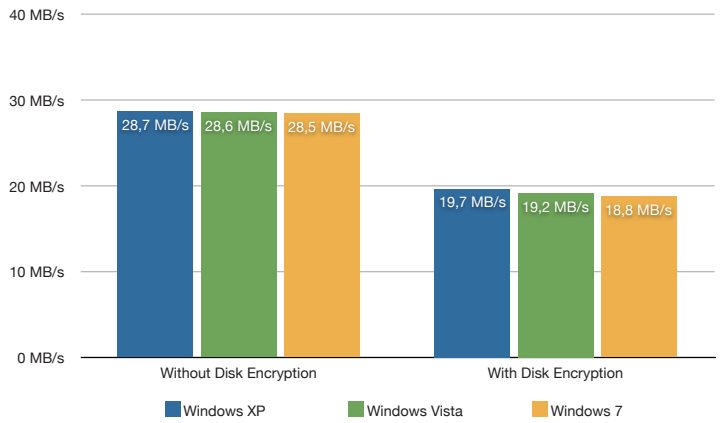
We observe that,

- Computer 1's read speed is approximately 29 MB/s when operating without disk encryption and approximately 19 MB/s when employing full system disk encryption, yielding a decrease of approximately *−35 %*.
- Computer 2's read speed is approximately 55 MB/s when operating without disk encryption and approximately 41 MB/s when employing full system disk encryption, yielding a decrease of approximately *−26 %*.
- Computer 3's read speed is approximately 94 MB/s when operating without disk encryption and approximately 66 MB/s when employing full system disk encryption, yielding a decrease of approximately *−30 %*.

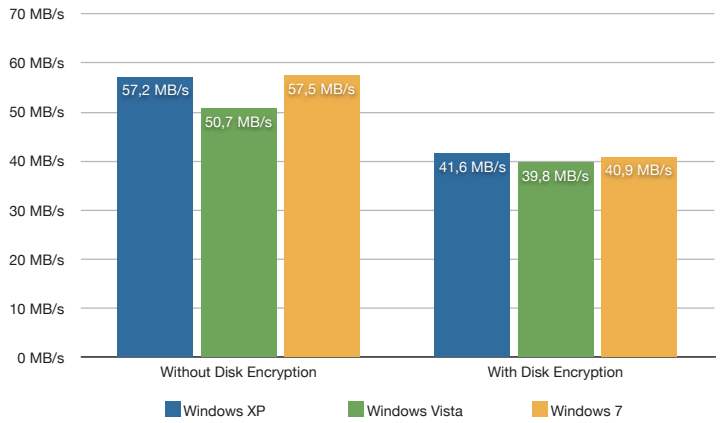
Also in this case we take notice that the percent wise change in speed is different for each computer. The gap between the highest percent wise decrease and lowest percent wise decrease is substantial also in this case, yielding approximately −35 % and approximately −26 %, respectively.

But unlike in the case of write speed, we do not recognize any special pattern. In fact, the computer with the disk that is measured to have the lowest read speed without disk encryption (i.e. computer 1) is the one experiencing the highest percent wise decrease when full system disk encryption is employed.

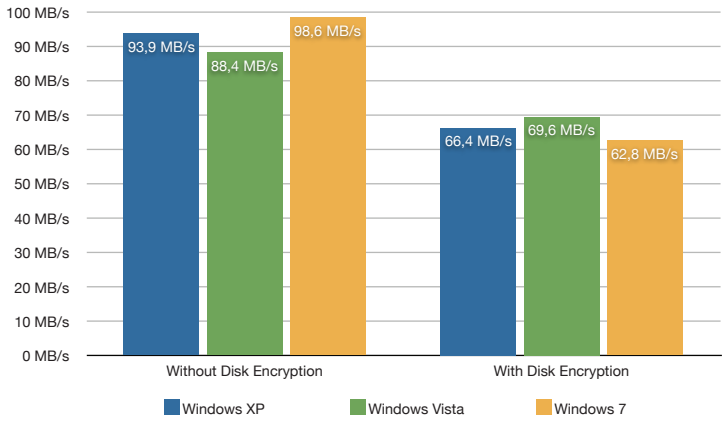
**Summary.** Our measurements show that the percent wise decrease in read speed is approximately −30 % (±5 %) when using full system disk encryption.




**(a)** *Average read speeds for computer 1 during benchmark*




**(b)** *Average read speeds for computer 2 during benchmark*




**(c)** *Average read speeds for computer 2 during benchmark*

**Figure 6.2:** *Average read speeds during benchmark for (a) computer 1, (b) computer 2, and (c) computer 3 using different operating systems.*

### 6.1.3 CPU Usage

Table 6.3 and Figure 6.3 shows the average CPU usage, both without and with disk encryption for each of the operating systems on the different computers. In addition, the numerical and percent wise change in each case is also listed.

| | **CPU usage** | | **Difference** | |
|---|---|---|---|---|
| | *Without disk encryption* | *With disk encryption* | *Numerical* | *Percent* |
| **Computer 1** | | | | |
| *Windows XP* | 1.1 % | 27.1 % | **26.0 %** | 2460.4 % |
| *Windows Vista* | 1.8 % | 24.5 % | **22.7 %** | 1278.1 % |
| *Windows 7* | 1.3 % | 22.0 % | **20.7 %** | 1659.2 % |
| **Computer 2** | | | | |
| *Windows XP* | 1.8 % | 52.2 % | **50.4 %** | 2863.6 % |
| *Windows Vista* | 2.8 % | 53.3 % | **50.5 %** | 1803.9 % |
| *Windows 7* | 2.2 % | 51.2 % | **49.0 %** | 2227.7 % |
| **Computer 3** | | | | |
| *Windows XP* | 2.0 % | 45.8 % | **43.8 %** | 2247.2 % |
| *Windows Vista* | 2.5 % | 50.3 % | **47.8 %** | 1913.6 % |
| *Windows 7* | 2.3 % | 53.5 % | **51.2 %** | 2214.7 % |

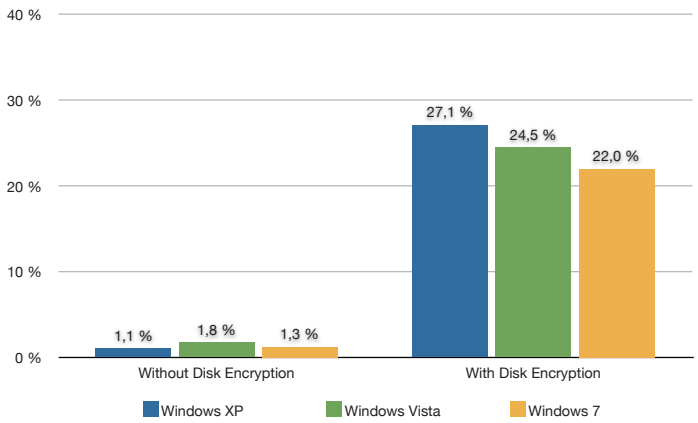
**Table 6.3:** *Average CPU usage during benchmark*

We observe that,

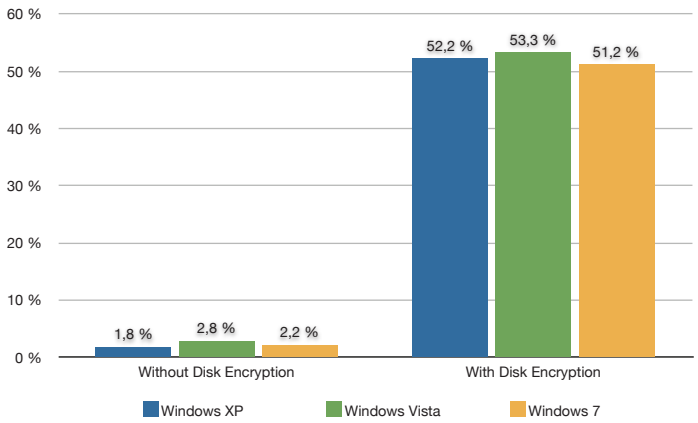
- Computer 1's CPU usage is approximately 2 % when no disk encryption was present, and went up to approximately 25 % when disk encryption was employed, yielding a numerical change of about *23 %*.
- Computer 2's CPU usage is approximately 2 % when no disk encryption was present, and went up to approximately 53 % when disk encryption was employed, yielding a numerical change of about *51 %*.
- Computer 3's CPU usage is approximately 2 % when no disk encryption was present, and went up to approximately 50 % when disk encryption was employed, yielding a numerical change of about *48 %*.

The obvious case point to be made from our observations is that while computer 2 and 3 experienced a numerical increase in CPU usage of approximately 50 %, computer 1 only experienced a numerical increase of 23 %. This is a major difference, and we have no definitive explanation for this. But, there is one discrepancy between the disks, namely that computer 1's disk is a 5,400rpm disk, while the two other computers have 7,200rpm disks. But again, since correlation does not necessarily imply causation, we cannot say for sure that this is the reason.

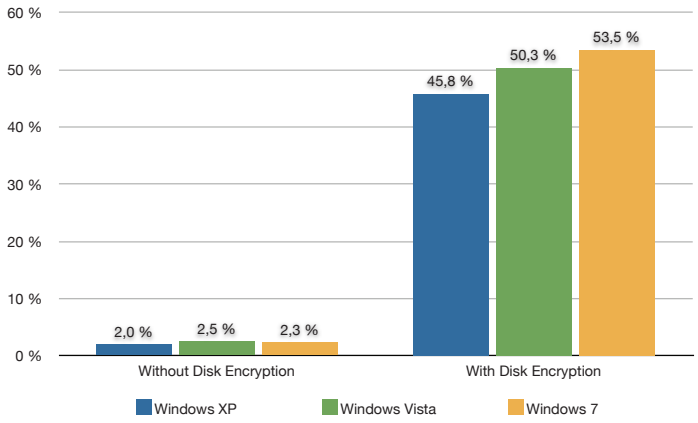
**Summary.** Our measurements show that disk operations (i.e. the benchmarking test) that without disk encryption took about 2 % of the CPU's time, takes up to approximately 50 % of the CPU's time when using full system disk encryption.




**(a)** *Average CPU usage of computer 1 during benchmark*




**(b)** *Average CPU usage of computer 2 during benchmark*




**(c)** *Average CPU usage of computer 3 during benchmark*

**Figure 6.3:** *Average CPU usage during benchmark for (a) computer 1, (b) computer 2, and (c) computer 3 using different operating systems.*

## 6.2 Analysis

In light of the results obtained, we see that applying XTS-AES-256 full system disk encryption does in fact noticeably affect a computers hard disk performance. All security mechanisms yield some performance decrease, which in the case of disk encryption is a decrease in transfer speed and increase in CPU usage during disk operations. A performance decrease was indeed expected, the question was more of which magnitude.

We recall from the previous sections that the decrease in write speed and read speed when introducing XTS-AES-256 full system disk encryption was measured to be up to $-35$ % (average for all operating systems). Whilst the CPU usage results showed that disk operations (i.e. the benchmark test) that used approximately 2 % of the CPU resources when no disk encryption was present, took approximately up to 50 % when full system disk encryption was employed.

### 6.2.1 Causality

Although the cause to the performance decrease might seem obvious, it should nevertheless be formally addressed. We believe that the performance decrease is caused by the overhead encountered when TrueCrypt transparently encrypts and decrypts all the data that goes to and from the disk, respectively. This process will obviously require the computational power of the CPU since the ciphering is performed in software. This in turn yields a decrease in transfer speed (due to the additional layer the data must pass through) and increase in CPU usage (due to the increased need for computational power).

### 6.2.2 Possible Consequences

Even though a decrease in disk speed of up to $-35$ % at best will not be noticed at all, the increase in CPU usage during disk operations is substantial. For simple operations like surfing the web, using word processors or spreadsheets, listening to music etc., the performance decrease (in transfer speed and available CPU power) might not yield any noticeable performance degradation. But, for resource intensive operations like high-resolution video editing, high-end computer gaming, and high-speed data transmissions this is surely unfavorable.

Other consequences that may or may not be a result of increased CPU usage during disk operations follows:

- More overall power consumption, which in the case of laptops might in turn result in a reduction in battery life time.
- Less computational power available for the user to utilize in his or hers actual work. This makes multitasking more challenging, and might even make certain programs close to non-respondent.
- A reduction in available CPU power (i.e. a degradation of the computer performance) may simply make the average user unwilling to use disk encryption.

When all is said and done, it is ultimately the perceived performance that determines whether the performance decrease is acceptable or not. This however, requires one to define and additionally assess the Quality of Experience (see Figure 6.4) in each test case, which is a qualitative research aspect not covered by our performance benchmark.

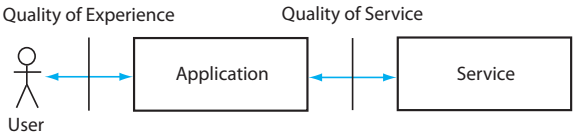



**Figure 6.4:** *Quality of Experience may be defined as the Quality of Service as perceived by the end-user.*

### 6.2.3 Limitations and Sources of Error

In order to comply with correct scientific procedure, we address limitations and possible sources of error in the following paragraphs.

Limitations in our performance benchmark:

- Due to time constraints and limited access to computer hardware, we were only able to perform benchmarking of three computers (and their respective hard disks). We believe that testing with more hardware is preferable to even more precisely assess the performance penalty that can be expected when introducing full system disk encryption.
- We chose to benchmark system disks, i.e. disks that the operating system resides on. Another approach would have been to benchmark the performance decrease experienced by non-system disks (i.e. full disk encryption, not full system disk encryption). Although the latter would also provide useful data, we had to choose one due to time limitations.
- We could have focused merely on one operating system, and thus had the time to perform even more tests (i.e. increased the sample size). The latter would further increase the reliability and validity of our data. However, we decided to include Windows XP, Windows Vista, and Windows 7 (Beta) to provide diversity.

Possible sources of error in our performance benchmark:

- The TrueCrypt daemon that transparently ciphers all data going to and from the disk might be subject to further optimization, i.e. improvements to minimize non-cryptographic overhead [46].
- The measurement precision of HD Tune Pro is another possible source of error. As specified in section 5.3.1, we chose a block size of 64 KB during the speed measurements, meaning that the speeds that are measured tell us how fast data is written or read when the file operations is done on data blocks of 64 KB in size. Although this was the recommended setting [43], we cannot exclude the possibility that choosing another block size would have yielded different results.

## 6.3 Further Work

Our performance benchmark of XTS-AES is, to our knowledge, the only scientific work one of its kind. We believe that assessing the real-world performance of XTS-AES is very important with respect to its viability and chances for widespread industry adoption. In the following paragraphs we indicate further work within the realm of performance testing of software-based implementations of XTS-AES that is yet to be done.

#### Systematic testing

The best overall picture of XTS-AES' affect on hard disks calls for systematic testing with respect to different brands, disk sizes (e.g., 60 GB, 100 GB, 250 GB, 500 GB), and speeds (e.g., 5,400rpm, 7,400rpm, and 10,000rpm disks). At the same time systematically varying the available computational power is also of utmost interest. The latter is important since software-based encryption solutions utilize the computational power of the computer using it.

#### Quality of Experience

The perceived performance when employing disk encryption solutions, i.e. the Quality of Experience, is also a possible subject for further research. Although a quantitative research of this nature is subjective, the results might very well be valuable for vendors of storage security products.

#### Other modes of operations for disk encryption

Real-world comparison with other modes of operations for disk encryption is also a point of great interest. This will however require a disk encryption application that is able to choose both narrow- and wide-block cipher modes. To our knowledge, no disk encryption software at the time of writing features "out-of-the-box" support for wide-block cipher modes.

#### Hard disks versus Flash disks

Since flash disks are becoming more and more commonly used as system disks, it would be interesting to also assess the prospective performance decrease in disks of this type. One of the most intriguing questions is probably whether the inherent high-speed nature of flash disks will completely choke the CPU when used with software-based encryption solutions or not.

#### Laptop battery time penalty

Another very practical and interesting test is how software-based disk encryption affects the battery time of a laptop. To test our suspicion about the battery time penalty, we did a very superficial test on computer 1 by playing a 720p video file in a loop both when employing disk encryption and not. This showed us that the battery time without disk encryption was approximately 64 minutes, but sunk to approximately 41 minutes when full system disk encryption was employed. For the battery time penalty to be assessed properly, one should preferably perform various tests on a wide variety of laptop and battery types.

# Chapter 7

# Conclusion

*I may not have gone where I intended to go, but I think I have ended up where I needed to be.*

**Douglas Adams**

Until IEEE Std 1619 was published in April 2008, there was no standardized or explicitly recommended mode of operation for disk encryption applications. Although not especially suited, ECB and CBC are still commonly used cipher modes in disk encryption solutions. But with the introduction of XTS-AES, the cryptographic industry now has a recommended narrow-block encryption mode explicitly designed for disk encryption usage.

The overall goal of this thesis was to provide a thorough examination of XTS-AES, describing both its security and real-world performance.

When reviewing the security of XTS-AES we observed that it is very hard to find exploitable similarities in XTS ciphertext of 128-bit. In fact, to even have a 1 % success probability of being able to distinguish XTS ciphertext from pseudo random permutations (i.e. birthday attack); one must be in possession of at least 32 exabyte of data encrypted with the same key. We further found that the only attacks applicable to XTS-AES at this time are generic attacks (that may be applied to any cryptosystem) or attacks that rely on the transparent narrow-block nature of XTS-AES. Thus, we have showed that XTS-AES provides a strong security guarantee as long as an adversary has limited access to ciphertext, the applications using the plaintext is able to detect malleability attacks, and the security scheme using XTS-AES has other security mechanisms in place to prevent generic attacks like exhaustive key search attack, dictionary attack, and cold boot attack.

Our unique real-world performance benchmark was conducted using TrueCrypt, which at the time of writing is the only application supporting XTS-AES and full system disk encryption. We measured the performance attributes; write speed, read speed and CPU usage during disk operations, both before and after employing full system XTS-AES-256 disk encryption. Our research showed that

the decrease in write and read speed was up to −35 % for computers running Windows (average for Windows XP SP2, Windows Vista SP1, and Windows 7 Beta). Whilst the CPU usage results showed that disk operations that used approximately 2 % of the CPU resources when no disk encryption was present, took approximately up to 50 % when full system disk encryption was employed. Thus, our findings imply that using a software-implementation of XTS-AES noticeably affects not only the write and read speed to disk, but also the available CPU power.

The industry adoption of XTS-AES is starting to pick up, and will likely flourish as a consequence of NIST very recently having accepted XTS as an approved mode of operation for AES. This strengthens the need for further systematic testing in order to even more precisely assess XTS-AES' performance impact on storage devices.

# References


[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. *A Concrete Security Treatment of Symmetric Encryption.* Proceedings 38th Annual Symposium on Foundations of Computer Science, FOCS'97. IEEE Computer Society, 1997.

[CC03] IEEE Computer Society: Storage Systems Standards Committee and Information Assurance Standards Committee. *Draft Proposal for Tweakable Wide-block Encryption.* March 2003. `https://siswg.net/docs/EME-AES-03-22-2004.pdf`.

[CC04] IEEE Computer Society: Storage Systems Standards Committee and Information Assurance Standards Committee. *Draft Proposal for Tweakable Narrow-block Encryption.* August 2004. `https://siswg.net/docs/LRW-AES-10-19-2004.pdf`.

[CC07] IEEE Computer Society: Storage Systems Standards Committee and Information Assurance Standards Committee. *IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices – IEEE Std 1619-2007.* December 2007. `http://ieeexplore.ieee.org/servlet/opac?punumber=4493431`.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. *Non-Malleable Cryptography.* March 2000.

[DR03] Joan Daemen and Vincent Rijmen. *A Specification for Rijndael, the AES Algorithm.* September 2003. `http://fp.gladman.plus.com/cryptography_technology/rijndael/aes.spec.311.pdf`.

[EFD08a] Mohamed Abo El-Fotouh and Klaus Diepold. *The Analysis of Windows Vista Disk Encryption Algorithm.* 2008.

[EFD08b] Mohamed Abo El-Fotouh and Klaus Diepold. *A New Narrow Block Mode of Operations for Disk Encryption.* September 2008. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4627074`.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications, Volume 1.* Wiley, 1968.

[Fer06] Niels Ferguson. *AES-CBC + Elephant diffuser. A Disk Encryption Algorithm for Windows Vista.* August 2006. `http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a0be4bbb36e/BitLockerCipher200608.pdf`.

[Fru05] Clemens Fruhwirth. *New Methods in Hard Disk Encryption*. 2005.

[HC03] Jim Hughes and Jack Cole. *Security in Storage*. January 2003. `http://www.msstc.org/cole/security-in-storage-200301.pdf`.

[HR03a] Shai Halevi and Phillip Rogaway. *A Parallelizable Enciphering Mode*. June 2003. `http://seclab.cs.ucdavis.edu/papers/eme.pdf`.

[HR03b] Shai Halevi and Phillip Rogaway. *A Tweakable Enciphering Mode*. June 2003. `http://seclab.cs.ucdavis.edu/papers/cmc.pdf`.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Josept A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. *Lest We Remember: Cold Boot Attacks on Encryption Keys*. February 2008. `http://citp.princeton.edu/pub/coldboot.pdf`.

[JBSK01] Borka Jerman-Blazic, Wolfgang S. Schneider, and Tomaz Klobucar. *Advanced Security Technologies in Networking*. Ios Pr Inc, 2001.

[Jou03] Antoine Joux. *Cryptanalysis of the EMD Mode of Operation*. May 2003. `http://www.ssi.gouv.fr/fr/sciences/fichiers/lcr/jo03.pdf`.

[Kah97] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Simon & Schuster, 1997.

[Ker83] Auguste Kerckhoffs. *La cryptographie militaire*, volume 9. Journal des sciences militaires, 1883. `http://www.petitcolas.net/fabien/kerckhoffs/#english`.

[LM08] Moses Liskov and Kazuhiko Minematsu. *Comments on XTS-AES*. September 2008. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/XTS_comments-Liskov_Minematsu.pdf`.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. *Tweakable Block Ciphers*. August 2002. `http://www.cs.berkeley.edu/~daw/papers/tweak-crypto02.pdf`.

[MF04] David A. McGrew and Scott Fluhrer. *The Extended Codebook (XCB) Mode of Operation*. October 2004. `http://eprint.iacr.org/2004/278.pdf`.

[Mil56] George Miller. *The Magical Number Seven, Plus or Minus Two*, volume 63. The Psychological Review, 1956.

[Min07] Kazuhiko Minematsu. *Improved Security Analysis of XEX and LRW Modes*. 2007. `http://www.springerlink.com/content/437264702kn51263/fulltext.pdf`.

[MvOV01] Alfred J. Menezes, Paul C. van Oorchot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[oSN] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards (FIPS) Publication (PUB) 197 - Specification for the Advanced Encryption Standard (AES).* `http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[oSN07] National Institute of Standards and Technology (NIST). *Recommendation for Key Management - Part 1: General (Revised) - SP800-57.* March 2007. `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf`.

[Rob95] Matt Robshaw. *Block Ciphers.* RSA Laboratories Technical Report TR-601, August 1995.

[Rog04] Phillip Rogaway. *Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC.* September 2004. `http://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf`.

[Sch06] Bruce Schneier. *Applied Cryptography.* Pearson Prentice Hall, 2006.

[Sha49] Claude Shannon. *Communication Theory of Secrecy Systems.* Bell Systems Technical Journal, No. 4, 1949.

[Sta06] William Stallings. *Cryptography and Network Security.* Pearson Prentice Hall, 2006.

[Tan02] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, 2002.

[vT06] Henk C.A. van Tilborg. *Encyclopedia of Cryptography and Security.* Springer-Verlag New York Inc., 2006.

# Web Resources


[1] Apple. Mac OS X Security, Visited: 03-04-2009. http://http://images.apple.com/macosx/pdf/MacOSX_Leopard_Security_TB.pdf.

[2] IEEE Standards Association. IEEE Approves Standards for Data Encryption, Visited: 27-01-2009. http://standards.ieee.org/announcements/StdsForEncryption.html.

[3] Matt Ball. Overview of the IEEE Security in Storage Working Group (SISWG) - As presented at Crypto 2008, Visited: 10-02-2009. https://siswg.net/index2.php?option=com_docman&task=doc_view&gid=137&Itemid=41.

[4] Matthew Ball. SISWG P1619 Task Group Minutes 27-09-2006, Visited: 15-03-2006. https://siswg.net/index.php?option=com_content&task=view&id=91.

[5] BestCrypt. BestCrypt Software Family, Visited: 19-02-2009. http://www.jetico.com/bcrypt.htm.

[6] Jon Callas. PGP Blogs, CTO Corner - Jon Callas, Visited: 26-04-2009. http://blog.pgp.com/index.php/category/cto_corner/.

[7] National Security Agency (NSA) CNSS Secretariat. Fact Sheet - CNSS Policy No. 15, Fact Sheet No. 1 National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, Visited: 04-02-2009. http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf.

[8] Open Source Community. cryptmount - a utility for accessing encrypted filesystems, Visited: 14-02-2009. http://cryptmount.sourceforge.net/.

[9] Open Source Community. dm-crypt: a device-mapper crypto target, Visited: 14-02-2009. http://www.saout.de/misc/dm-crypt/.

[10] CrossCrypt. Open Source AES and TwoFish Linux compatible on the fly encryption for Windows XP and Windows 2000, Visited: 19-02-2009. http://scherrer.cc/crypt/.

[11] Sarah Dean. FreeOTFE, Visited: 13-02-2009. http://www.freeotfe.org/.

[12] Sarah Dean. FreeOTFE - Frequently Asked Questions, Visited: 13-02-2009. http://www.freeotfe.org/docs/FAQ.htm.

[13] DriveCrypt. Securstar, encryption software solutions, Visited: 19-02-2009. http://www.securstar.com/products_drivecrypt.php.

[14] Eric A. Hibbard. Storage Developer Conference 2008 - Status Report on Storage Security Initiatives, Visited: 18-02-2009. http://www.snia.org/events/storage-developer2008/presentations/tuesday/EricHibbard_StateOfStorageSecurityInitiatives.pdf.

[15] IEEE Security in Storage Working Group. P1619 Standard Architecture for Encrypted Shared Storage Media, Visited: 20-01-2009. https://siswg.net/index.php?option=com_content&task=view&id=38&Itemid=73.

[16] IEEE Security in Storage Working Group. P1619.1 Standard for Authenticated Encryption with Length Expansion for Storage Devices, Visited: 20-01-2009. https://siswg.net/index.php?option=com_content&task=view&id=38&Itemid=74.

[17] IEEE Security in Storage Working Group. P1619.2 Standard for Wide-Block Encryption for Shared Storage Media, Visited: 20-01-2009. https://siswg.net/index.php?option=com_content&task=view&id=37&Itemid=75.

[18] IEEE Security in Storage Working Group. P1619.3 Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data, Visited: 20-01-2009. https://siswg.net/index.php?option=com_content&task=view&id=38&Itemid=76.

[19] IEEE Security in Storage Working Group. IEEE 1619 SISWG email archive, Visited: 27-01-2009. http://grouper.ieee.org/groups/1619/email/.

[20] IEEE Security in Storage Working Group. IEEE 1619 SISWG Security in Storage Working Group, Visited: 27-01-2009. http://siswg.net/.

[21] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 3548 (Informational), July 2003. http://www.ietf.org/rfc/rfc3548.txt.

[22] RSA Laboratories. Frequently Asked Questions about Today's Cryptography, Visited: 31-02-2009. http://www.rsa.com/rsalabs/node.asp?id=2152.

[23] Microsoft. Explore the features: Bitlocker Drive Encryption, Visited: 24-03-2009. http://www.microsoft.com/windows/windows-vista/features/bitlocker.aspx.

[24] James Morris and David S. Miller. Crypto API for Linux, Visited: 14-02-2009. http://gondor.apana.org.au/~herbert/crypto/.

[25] Matthew V. Ball (Chair of IEEE SISWG). Follow-up to NIST's Consideration of XTS-AES as standardized by IEEE Std 1619-2007 (Draft 3 - April 12, 2009), Visited: 01-05-2009. https://siswg.net/index.php?option=com_docman&task=doc_download&gid=169&Itemid=41.

[26] Matthew V. Ball (Chair of IEEE SISWG). Email to NIST dated September 1, 2008, Visited: 15-03-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/XTS_comments-Ball.pdf.

[27] Matthew V. Ball (Chair of IEEE SISWG). Email to P1619 working group dated May 26, 2009 with subject: "NIST will accept XTS-AES (as defined by IEEE Std 1619-2007) as an Approved Mode of Operation", Visited: 27-05-2009. http://grouper.ieee.org/groups/1619/email/msg02579.html.

[28] National Institute of Standards and Technology (NIST). Key Management Guidelines SP800-57, August 2005. http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf and http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf.

[29] National Institute of Standards and Technology (NIST). Proposal to Extend CBC Mode By "Ciphertext Stealing", May 2007. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ciphertext%20stealing%20proposal.pdf.

[30] National Institute of Standards and Technology (NIST). Press Release 09-08-1999: NIST Announces Encryption Standard Finalists, Visited: 05-02-2009. http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=2001_register&docid=01-4886-filed.pdf.

[31] National Institute of Standards and Technology (NIST). Comments on the Proposal to Approve XTS-AES, Visited: 15-02-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/comments.html.

[32] National Institute of Standards and Technology (NIST). Request for Public Comment on XTS, Visited: 15-02-2009. http://csrc.nist.gov/groups/ST/documents/Request-for-Public-Comment-on_XTS.pdf.

[33] National Institute of Standards and Technology (NIST). Public Comments on the XTS-AES Mode, Visited: 15-03-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/collected_XTS_comments.pdf.

[34] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES) Questions and Answers, Visited: 27-01-2009. http://www.nist.gov/public_affairs/releases/aesq&a.htm.

[35] National Institute of Standards and Technology (NIST). Archived info: C S R C - Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard, FIPS-197, Visited: 27-01-2009. http://csrc.nist.gov/encryption/aes/.

[36] National Institute of Standards and Technology (NIST). Federal Information Processing Standards (FIPS) Publications, Visited: 29-02-2009. http://csrc.nist.gov/publications/PubsFIPS.html.

[37] National Institute of Standards and Technology. NIST Accepted Block Cipher Modes of Operation for Consideration, Visited: 23-01-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html.

[38] National Institute of Standards and Technology. NIST Approved Block Cipher Modes of Operation, Visited: 30-01-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html.

[39] Trusted Computing Group Press Release. Better Protection from Client to Data Center Made Possible With New Trusted Computing Group Storage Device Specifications, January 2009. http://www.trustedcomputinggroup.org/media_room/news/15.

[40] Philip Rogaway. Quick check on IP situation of XEX, September 2006. http://grouper.ieee.org/groups/1619/email/msg01309.html.

[41] Stephan Schreiber. Ext2 Installable File System, Visited: 14-02-2009. http://www.fs-driver.org/.

[42] Micheal Willett (Seagate). Request for Public Comment on XTS-AES, Visited: 15-03-2009. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/revised_XTS_comments-Seagate.pdf.

[43] EFD Software. HD Tune, Visited: 11-02-2009. http://www.hdtunepro.com.

[44] NeoScale Systems. Letter of Assurance to IEEE on IEEE Std 1619, March 2007. http://standards.ieee.org/db/patents/loa-1619-neoscale-08Mar2007.pdf.

[45] TrueCrypt. Free open-source disk encryption software for Windows Vista/XP, Mac OS X, and Linux, Visited: 01-02-2009. http://www.truecrypt.org/.

[46] TrueCrypt. Version History, Visited: 01-02-2009. http://www.truecrypt.org/docs/?s=version-history.

[47] TrueCrypt. Frequently Asked Questions, Visited: 04-02-2009. http://www.truecrypt.org/faq.php.

[48] TrueCrypt. Header Key Derivation, Salt, and Iteration Count, Visited: 08-03-2009. http://www.truecrypt.org/docs/?s=header-key-derivation.

[49] TrueCrypt. Random Number Generator, Visited: 08-03-2009. http://www.truecrypt.org/docs/?s=random-number-generator.

[50] TrueCrypt. Truecrypt Rescue Disk, Visited: 08-03-2009. http://www.truecrypt.org/docs/?s=rescue-disk.

[51] World Wide Web Consortium (W3C). XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002. http://www.w3.org/TR/xmlenc-core.

[52] World Wide Web Consortium (W3C). Definition of the XML document type declaration from Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation 16 August 2006. `http://www.w3.org/TR/REC-xml/#dt-doctype`.

[53] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation 16 August 2006. `http://www.w3.org/TR/REC-xml/`.

# Appendix A

# TrueCrypt 6.1a

This Appendix gives step-by-step instructions on how *full system disk encryption* was applied when the test case named `XTS-AES-256_Encryption` (described in section 5.4.2) was performed.

### Step 1

TrueCrypt 6.1a was downloaded from [45] and installed with default settings.

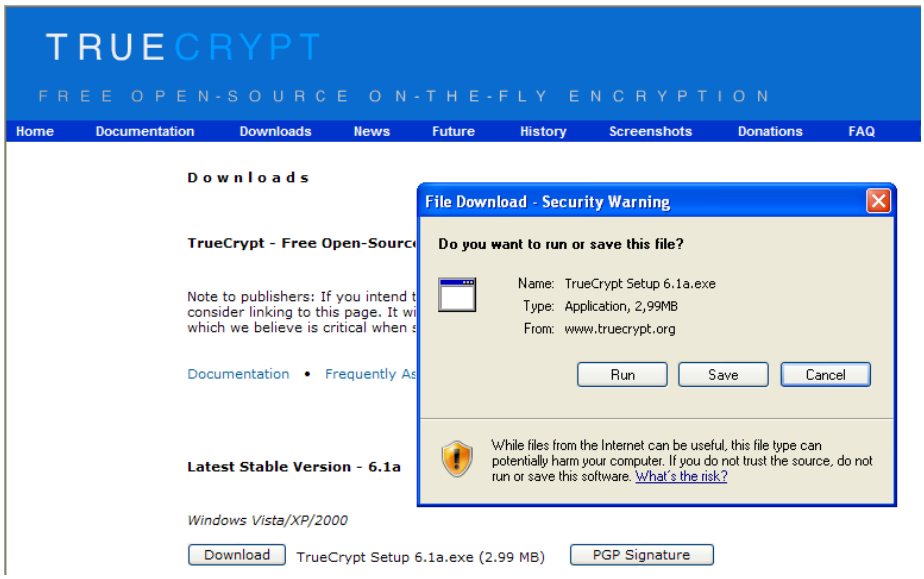



**Figure A.1:** *At the time of testing the latest stable version of TrueCrypt was 6.1a.*

### Step 2

The next step was to open TrueCrypt and choose *System* → *Encrypt System Partition/Drive* (as depicted in Figure A.2) to start the TrueCrypt Volume Creation Wizard.

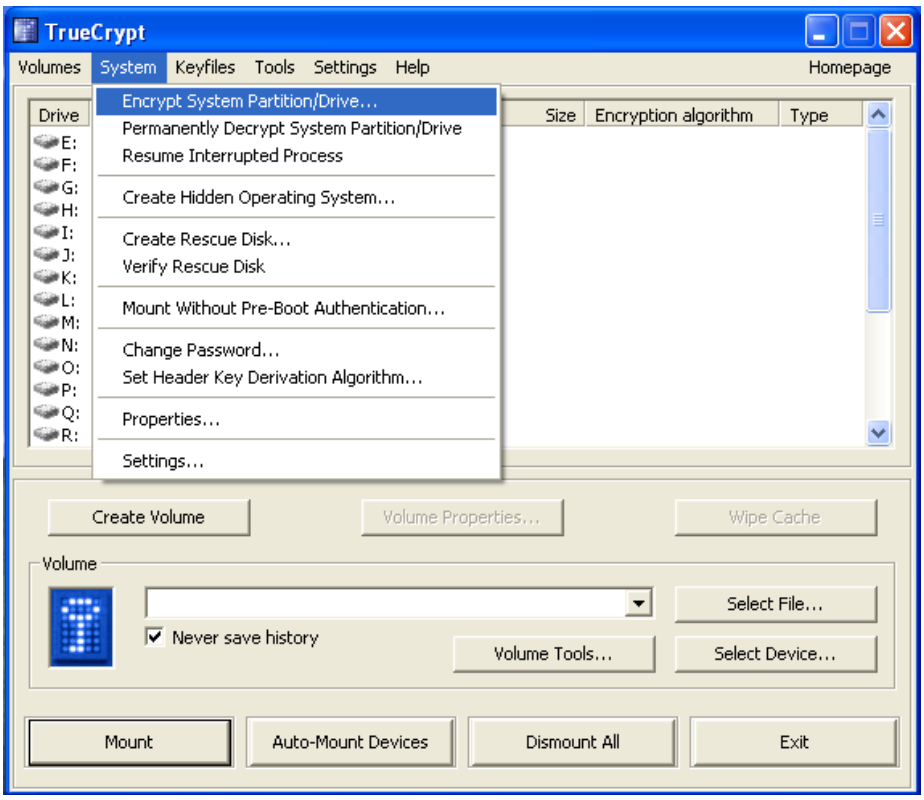



**Figure A.2:** *Full system disk encryption is initiated from the TrueCrypt main window.*

### Step 3

At this point TrueCrypt asks a couple of questions regarding the manner of operation, which we answered as follows:

- Type of System Encryption: *Normal*
- Area to Encrypt: *Encrypt the whole drive*
- Encryption of Host Protected Area: *Yes*
- Number of Operating Systems: *Single-boot*

### Step 4

After this, TrueCrypt presents the user with encryption algorithms to choose from. As depicted in Figure A.3, *AES* was selected, which uses the XTS mode of operation with a key size of 256 bits (XTS-AES-256).

Notice that you also can choose a hash algorithm at this point. This is used by the random number generator (as a pseudorandom mixing function), which is involved in the generation of the master key (AES key), secondary key (XTS key), salt, volume header key, and secondary header key [48, 49]. The default hash algorithm (i.e. RIPEMD-160) was selected.

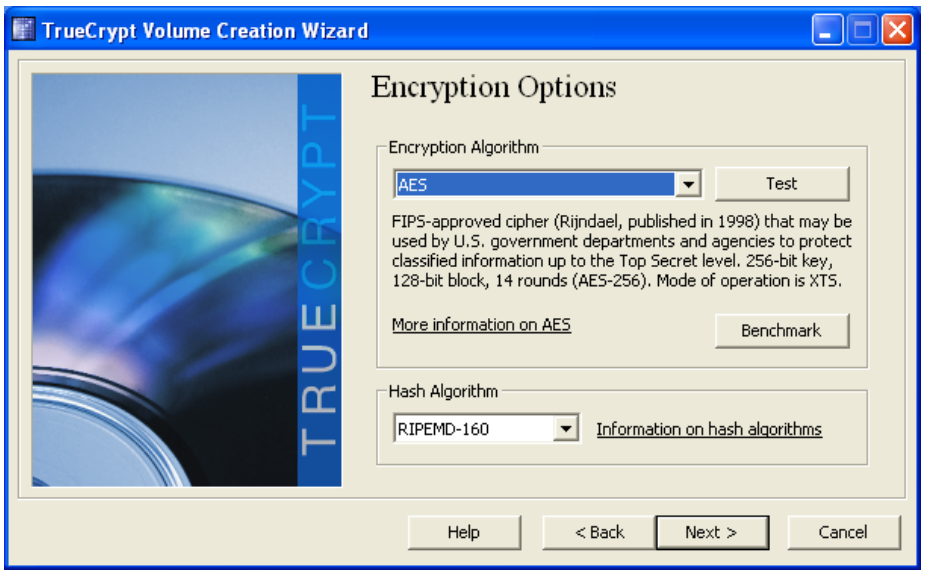



**Figure A.3:** *TrueCrypt features strong 256-bit encryption using XTS mode with either AES, Twofish or Serpent block cipher, or all three of them in cascade mode.*

### Step 5

The next step was to choose the password used for pre-boot authentication.

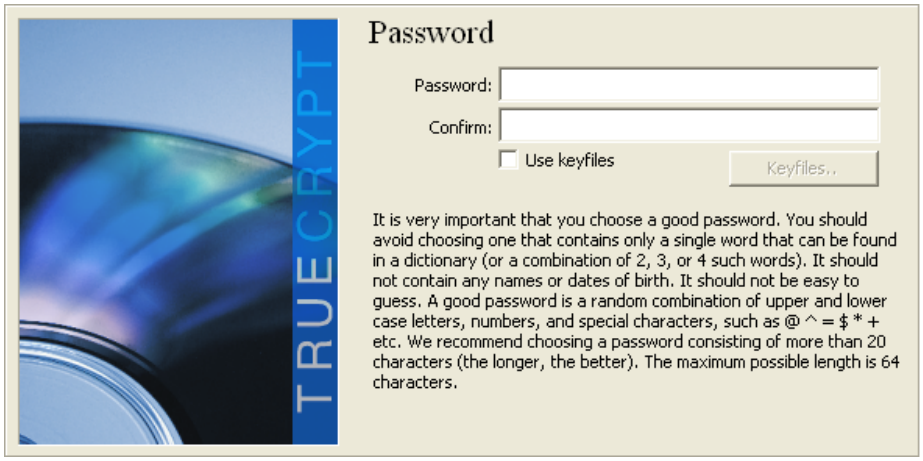



**Figure A.4:** *Notice that TrueCrypt also features the use of a keyfile, which is a file whose content is combined with a password.*

### Step 6

After choosing a password, TrueCrypt generates the necessary keys by using the password defined in the previous step and random data collected from your mouse cursor movement (see Figure A.5). When this process was complete, a new TrueCrypt window displayed the generated keys.

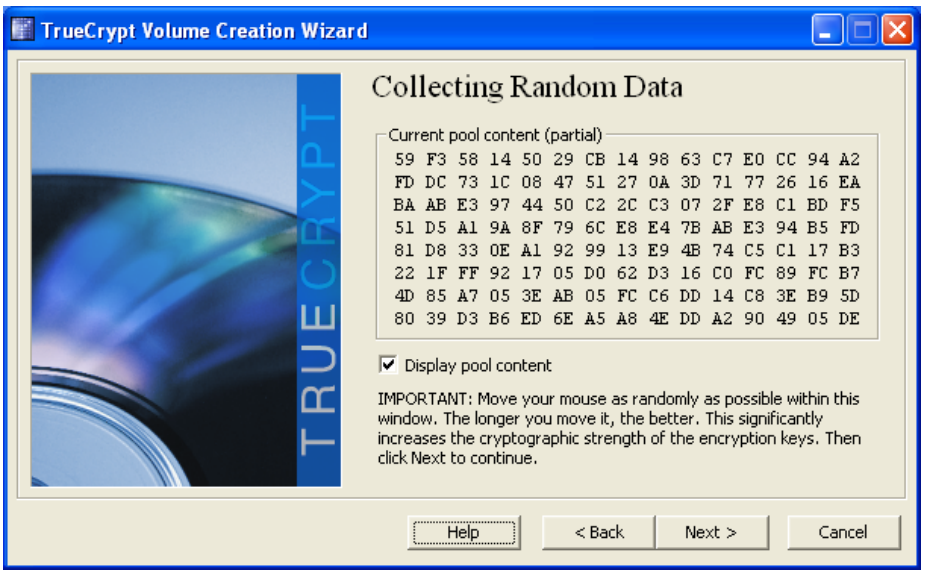



**Figure A.5:** *Mouse cursor movement inside the TrueCrypt window is used to collect random numbers.*

### Step 7

Before the process of full system disk encryption starts, TrueCrypt requires the user to create a TrueCrypt Rescue Disk. This is used if the TrueCrypt Boot Loader, header key, or other critical data gets damaged. More about the necessity and use of the TrueCrypt Rescue Disk is found at [50].

After the Rescue Disk image was created, it was written (burned) to an empty CD-R, and verified by TrueCrypt.


**Figure A.6:** *Notice that it is not possible to proceed without getting your Rescue Disk verified.*

### Step 8

At this point, only a few clicks away from the actual encryption process, TrueCrypt requires the user to choose a Wipe mode (see Figure A.7). This is done because overwritten data might be recovered using techniques such as magnetic force microscopy. This also applies to data that are overwritten with their encrypted form (which is exactly what TrueCrypt are about to do). For the testing purposes in this thesis, the Wipe mode was set to *None*.

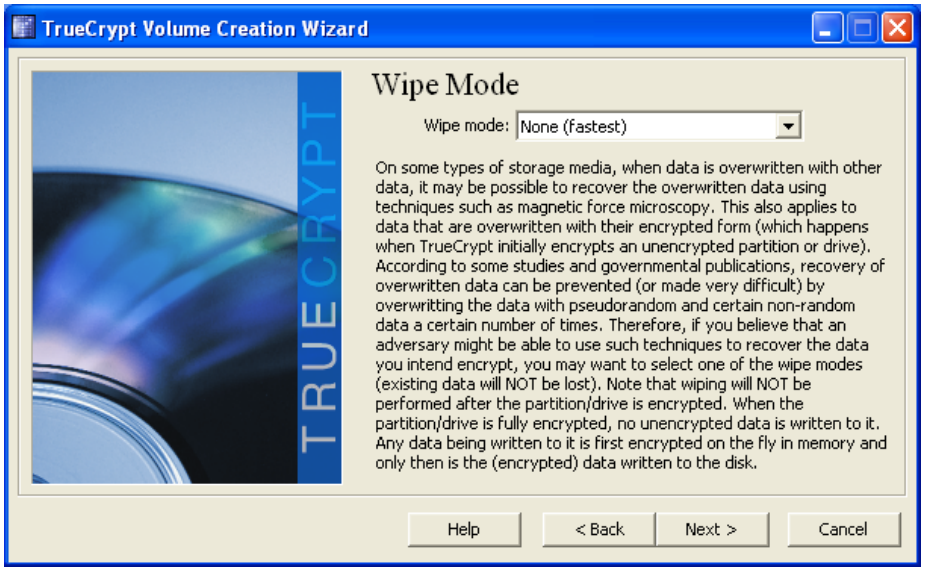



**Figure A.7:** *It is possible to choose between four wipe modes: None, 3-pass, 7-pass and 35-pass.*

### Step 9

Then, before the actual encrypting can start, TrueCrypt needs to verify that everything workings correctly. This was done by restarting and authenticating us to the TrueCrypt Boot Loader (see Figure A.8).

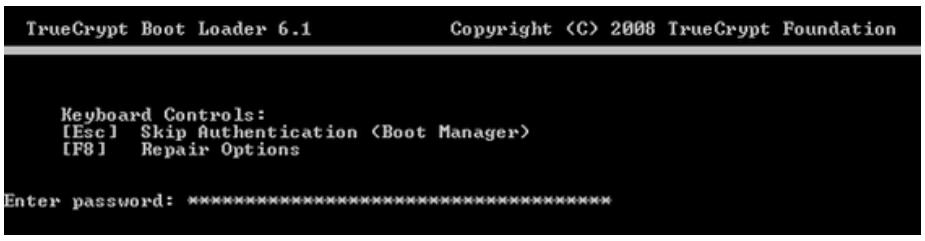



**Figure A.8:** *As of TrueCrypt version 6.1, it is possible to turn off all text in the pre-boot authentication screen, preventing an adversary that is watching you start your computer from knowing that TrueCrypt is in use.*

### Step 10

After Windows restarted, and the pretest was performed successfully, the actual encryption started (depicted in Figure A.9).


**Figure A.9:** *Notice that it is possible to pause the full system disk encryption process at any time.*

After the encryption process is complete, TrueCrypt never saves any decrypted data to the disk again – it only stores it temporarily in computer memory. From this point on, data is automatically encrypted or decrypted right before it is written or read, completely transparent to the user.

# Appendix B

# Benchmark Results

This Appendix presents the exhaustive result sets with corresponding statistics; arithmetic average, standard deviation, standard error, and 95 % confidence intervals.

The table below gives an overview of the different computers and operating systems along with the page number on which the results is listed; even page numbers show the results when no disk encryption was present (i.e. the test case `No_Encryption`) and odd page numbers show the results when the disk employed transparent XTS-AES-256 full system disk encryption (i.e. the test case `XTS-AES-256_Encryption`).

| | Operating System | Pages |
|---|---|---|
| **Computer 1** | Windows XP | 102 – 103 |
| | Windows Vista | 104 – 105 |
| | Windows 7 | 106 – 106 |
| **Computer 2** | Windows XP | 108 – 109 |
| | Windows Vista | 110 – 111 |
| | Windows 7 | 112 – 113 |
| **Computer 3** | Windows XP | 114 – 115 |
| | Windows Vista | 116 – 117 |
| | Windows 7 | 118 – 119 |

# B.1 Computer 1 using Windows XP

## Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 27.46 MB/s | 28.95 MB/s | 1.10 % |
| *#2* | 27.11 MB/s | 29.66 MB/s | 1.10 % |
| *#3* | 26.52 MB/s | 29.15 MB/s | 1.00 % |
| *#4* | 27.17 MB/s | 28.42 MB/s | 1.10 % |
| *#5* | 27.00 MB/s | 28.96 MB/s | 1.00 % |
| *#6* | 28.31 MB/s | 28.79 MB/s | 1.00 % |
| *#7* | 27.19 MB/s | 28.43 MB/s | 1.20 % |
| *#8* | 27.20 MB/s | 28.60 MB/s | 1.00 % |
| *#9* | 27.17 MB/s | 27.83 MB/s | 1.10 % |
| *#10* | 26.81 MB/s | 28.41 MB/s | 1.00 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **27.2 MB/s** | **28.7 MB/s** | **1.1 %** |
| *Standard deviation* | 0.47 | 0.50 | 0.07 |
| *Standard error* | 0.15 | 0.16 | 0.02 |
| *95% Confidence interval* | [26.87,27.53] | [28.37,29.07] | [1.05,1.15] |

**Table B.1:** *Disk performance measurements for computer 1 running Windows XP without disk encryption.*

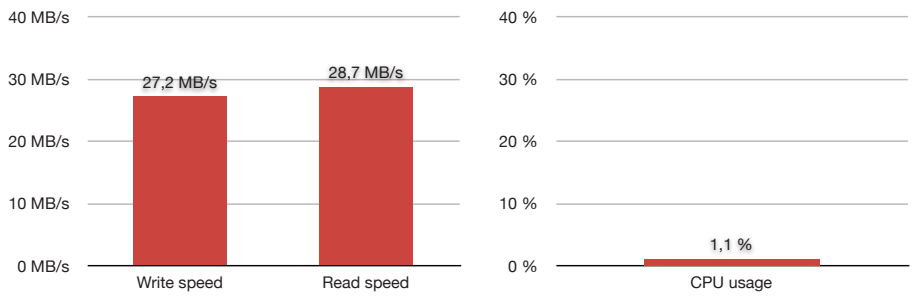



**Figure B.1:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows XP without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 21.24 MB/s | 21.26 MB/s | 26.90 % |
| *#2* | 21.21 MB/s | 20.94 MB/s | 27.50 % |
| *#3* | 21.12 MB/s | 20.61 MB/s | 27.30 % |
| *#4* | 21.20 MB/s | 17.35 MB/s | 27.50 % |
| *#5* | 21.19 MB/s | 20.60 MB/s | 27.00 % |
| *#6* | 21.26 MB/s | 17.08 MB/s | 26.80 % |
| *#7* | 21.19 MB/s | 20.71 MB/s | 27.30 % |
| *#8* | 21.19 MB/s | 20.49 MB/s | 26.90 % |
| *#9* | 21.30 MB/s | 20.80 MB/s | 27.10 % |
| *#10* | 21.08 MB/s | 17.64 MB/s | 27.10 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **21.2 MB/s** | **19.7 MB/s** | **27.1 %** |
| *Standard deviation* | 0.06 | 1.67 | 0.25 |
| *Standard error* | 0.02 | 0.53 | 0.08 |
| *95% Confidence interval* | [21.15,21.24] | [18.57,20.92] | [26.92,27.28] |

**Table B.2:** *Disk performance measurements for computer 1 running Windows XP with disk encryption.*

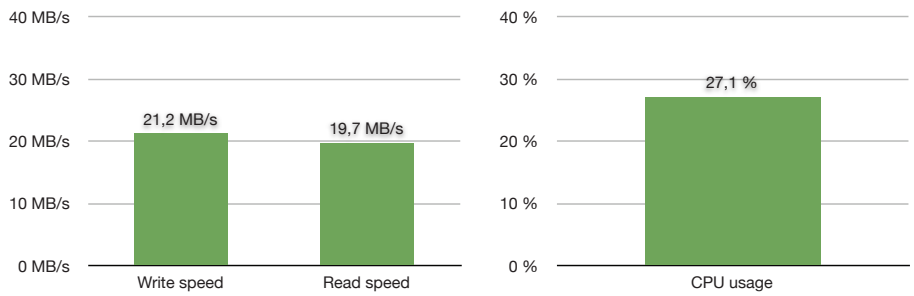



**Figure B.2:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows XP with disk encryption.*

## B.2 Computer 1 using Windows Vista

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 28.41 MB/s | 29.71 MB/s | 2.20 % |
| *#2* | 28.85 MB/s | 29.73 MB/s | 1.90 % |
| *#3* | 29.15 MB/s | 29.52 MB/s | 1.80 % |
| *#4* | 28.44 MB/s | 29.71 MB/s | 1.80 % |
| *#5* | 28.79 MB/s | 29.61 MB/s | 2.20 % |
| *#6* | 25.21 MB/s | 27.40 MB/s | 1.50 % |
| *#7* | 24.16 MB/s | 27.41 MB/s | 1.40 % |
| *#8* | 25.04 MB/s | 27.42 MB/s | 1.80 % |
| *#9* | 25.12 MB/s | 27.56 MB/s | 1.60 % |
| *#10* | 25.12 MB/s | 27.96 MB/s | 1.60 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **26.8 MB/s** | **28.6 MB/s** | **1.8 %** |
| *Standard deviation* | 2.03 | 1.12 | 0.27 |
| *Standard error* | 0.64 | 0.36 | 0.09 |
| *95% Confidence interval* | [25.40,28.26] | [27.81,29.39] | [1.61,1.99] |

**Table B.3:** *Disk performance measurements for computer 1 running Windows Vista without disk encryption.*

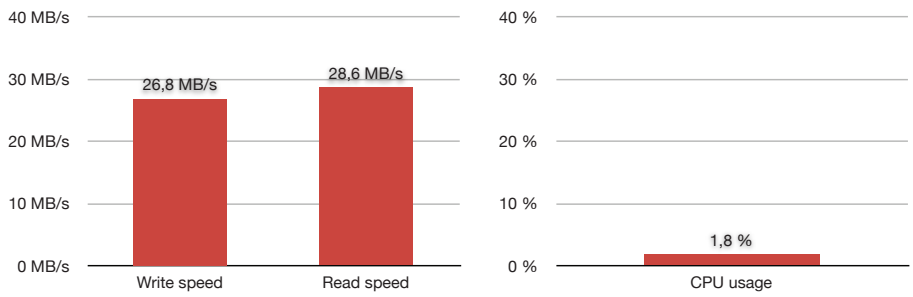



**Figure B.3:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows Vista without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 21.10 MB/s | 19.33 MB/s | 24.50 % |
| *#2* | 21.02 MB/s | 19.43 MB/s | 23.20 % |
| *#3* | 20.77 MB/s | 19.21 MB/s | 26.30 % |
| *#4* | 20.71 MB/s | 19.18 MB/s | 26.20 % |
| *#5* | 20.76 MB/s | 18.94 MB/s | 23.00 % |
| *#6* | 20.75 MB/s | 19.50 MB/s | 22.60 % |
| *#7* | 20.52 MB/s | 19.11 MB/s | 23.90 % |
| *#8* | 20.71 MB/s | 19.74 MB/s | 26.70 % |
| *#9* | 20.94 MB/s | 18.69 MB/s | 24.50 % |
| *#10* | 20.92 MB/s | 19.20 MB/s | 24.40 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **20.8 MB/s** | **19.2 MB/s** | **24.5 %** |
| *Standard deviation* | 0.17 | 0.30 | 1.45 |
| *Standard error* | 0.05 | 0.09 | 0.46 |
| *95% Confidence interval* | [20.70,20.94] | [19.03,19.44] | [23.48,25.52] |

**Table B.4:** *Disk performance measurements for computer 1 running Windows Vista with disk encryption.*

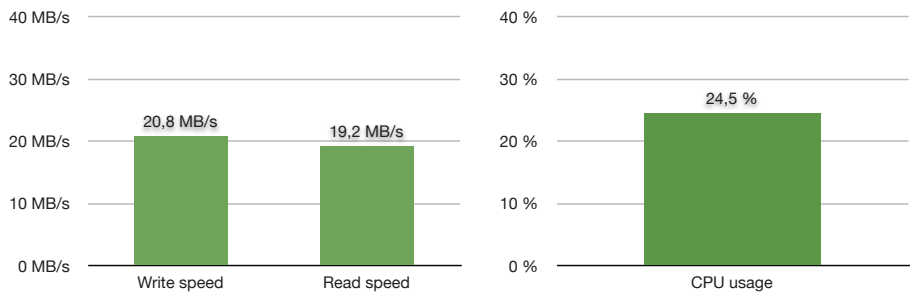



**Figure B.4:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows Vista with disk encryption.*

## B.3 Computer 1 using Windows 7

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 27.24 MB/s | 28.86 MB/s | 1.20 % |
| *#2* | 27.58 MB/s | 29.18 MB/s | 1.30 % |
| *#3* | 27.87 MB/s | 28.81 MB/s | 1.30 % |
| *#4* | 26.81 MB/s | 29.23 MB/s | 1.20 % |
| *#5* | 27.12 MB/s | 29.15 MB/s | 1.30 % |
| *#6* | 26.93 MB/s | 22.77 MB/s | 1.30 % |
| *#7* | 27.05 MB/s | 29.07 MB/s | 1.20 % |
| *#8* | 26.80 MB/s | 29.48 MB/s | 1.30 % |
| *#9* | 26.91 MB/s | 29.32 MB/s | 1.10 % |
| *#10* | 27.35 MB/s | 29.57 MB/s | 1.30 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **27.2 MB/s** | **28.5 MB/s** | **1.3 %** |
| *Standard deviation* | 0.35 | 2.04 | 0.07 |
| *Standard error* | 0.11 | 0.65 | 0.02 |
| *95% Confidence interval* | [26.92,27.41] | [27.11,29.98] | [1.25,1.35] |

**Table B.5:** *Disk performance measurements for computer 1 running Windows 7 without disk encryption.*

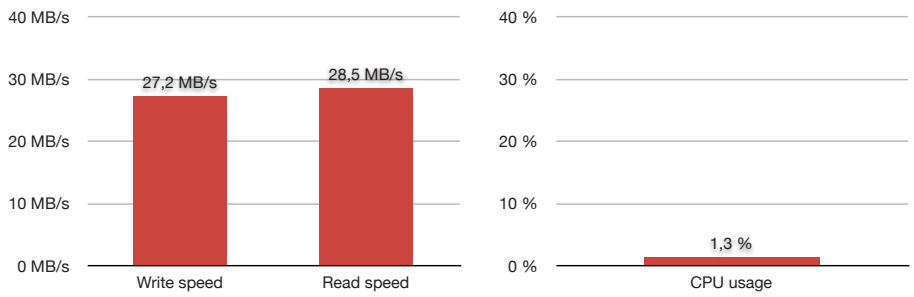



**Figure B.5:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows 7 without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 20.86 MB/s | 18.55 MB/s | 21.50 % |
| *#2* | 21.13 MB/s | 16.86 MB/s | 21.80 % |
| *#3* | 20.82 MB/s | 18.77 MB/s | 21.80 % |
| *#4* | 20.49 MB/s | 17.83 MB/s | 22.50 % |
| *#5* | 20.82 MB/s | 19.30 MB/s | 22.60 % |
| *#6* | 20.45 MB/s | 19.48 MB/s | 22.20 % |
| *#7* | 21.08 MB/s | 18.66 MB/s | 22.40 % |
| *#8* | 21.08 MB/s | 19.34 MB/s | 21.50 % |
| *#9* | 19.98 MB/s | 19.21 MB/s | 22.00 % |
| *#10* | 19.13 MB/s | 20.01 MB/s | 21.60 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **20.6 MB/s** | **18.8 MB/s** | **22.0 %** |
| *Standard deviation* | 0.62 | 0.91 | 0.41 |
| *Standard error* | 0.20 | 0.29 | 0.13 |
| *95% Confidence interval* | [20.15,21.02] | [18.16,19.44] | [21.71,22.29] |

**Table B.6:** *Disk performance measurements for computer 1 running Windows 7 with disk encryption.*

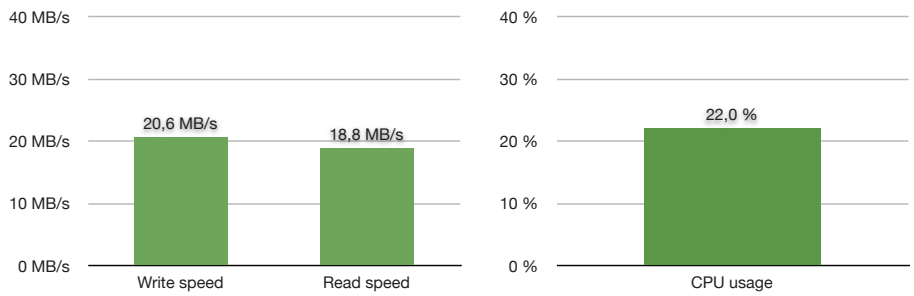



**Figure B.6:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 1 running Windows 7 with disk encryption.*

## B.4 Computer 2 using Windows XP

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 59.44 MB/s | 56.85 MB/s | 1.80 % |
| *#2* | 59.42 MB/s | 56.84 MB/s | 1.90 % |
| *#3* | 59.92 MB/s | 57.09 MB/s | 1.70 % |
| *#4* | 59.21 MB/s | 57.84 MB/s | 1.70 % |
| *#5* | 58.81 MB/s | 57.09 MB/s | 1.80 % |
| *#6* | 57.21 MB/s | 57.84 MB/s | 1.70 % |
| *#7* | 59.71 MB/s | 56.34 MB/s | 1.70 % |
| *#8* | 58.96 MB/s | 57.64 MB/s | 1.80 % |
| *#9* | 59.39 MB/s | 57.46 MB/s | 1.70 % |
| *#10* | 59.43 MB/s | 57.09 MB/s | 1.80 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **59.2 MB/s** | **57.2 MB/s** | **1.8 %** |
| *Standard deviation* | 0.76 | 0.48 | 0.07 |
| *Standard error* | 0.24 | 0.15 | 0.02 |
| *95% Confidence interval* | [58.62,59.68] | [56.87,57.55] | [1.75,1.85] |

**Table B.7:** *Disk performance measurements for computer 2 running Windows XP without disk encryption.*

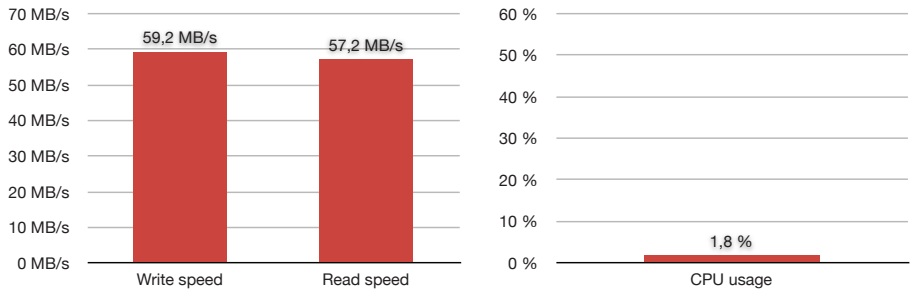



**Figure B.7:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows XP without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 42.11 MB/s | 42.29 MB/s | 52.20 % |
| *#2* | 42.13 MB/s | 42.09 MB/s | 51.90 % |
| *#3* | 42.04 MB/s | 41.91 MB/s | 52.10 % |
| *#4* | 42.13 MB/s | 41.49 MB/s | 52.40 % |
| *#5* | 42.11 MB/s | 40.86 MB/s | 52.20 % |
| *#6* | 42.12 MB/s | 40.01 MB/s | 52.30 % |
| *#7* | 42.09 MB/s | 42.97 MB/s | 51.80 % |
| *#8* | 42.11 MB/s | 41.82 MB/s | 52.40 % |
| *#9* | 42.10 MB/s | 40.95 MB/s | 52.40 % |
| *#10* | 42.11 MB/s | 41.91 MB/s | 51.90 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **42.1 MB/s** | **41.6 MB/s** | **52.2 %** |
| *Standard deviation* | 0.02 | 0.84 | 0.23 |
| *Standard error* | 0.01 | 0.27 | 0.07 |
| *95% Confidence interval* | [42.09,42.12] | [41.04,42.22] | [52.04,52.36] |

**Table B.8:** *Disk performance measurements for computer 2 running Windows XP with disk encryption.*

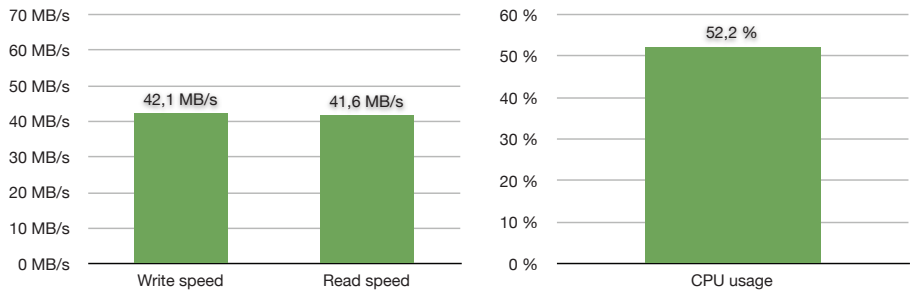



**Figure B.8:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows XP with disk encryption.*

## B.5 Computer 2 using Windows Vista

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 46.98 MB/s | 51.93 MB/s | 2.90 % |
| *#2* | 48.78 MB/s | 50.83 MB/s | 2.40 % |
| *#3* | 49.12 MB/s | 51.93 MB/s | 2.60 % |
| *#4* | 45.62 MB/s | 46.19 MB/s | 2.70 % |
| *#5* | 54.86 MB/s | 50.71 MB/s | 2.80 % |
| *#6* | 55.17 MB/s | 49.18 MB/s | 2.60 % |
| *#7* | 53.62 MB/s | 50.27 MB/s | 3.10 % |
| *#8* | 52.50 MB/s | 51.67 MB/s | 3.10 % |
| *#9* | 52.24 MB/s | 52.14 MB/s | 3.00 % |
| *#10* | 55.58 MB/s | 52.40 MB/s | 2.80 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **51.4 MB/s** | **50.7 MB/s** | **2.8 %** |
| *Standard deviation* | 3.58 | 1.88 | 0.23 |
| *Standard error* | 1.13 | 0.59 | 0.07 |
| *95% Confidence interval* | [48.92,53.97] | [49.40,52.05] | [2.64,2.96] |

**Table B.9:** *Disk performance measurements for computer 2 running Windows Vista without disk encryption.*

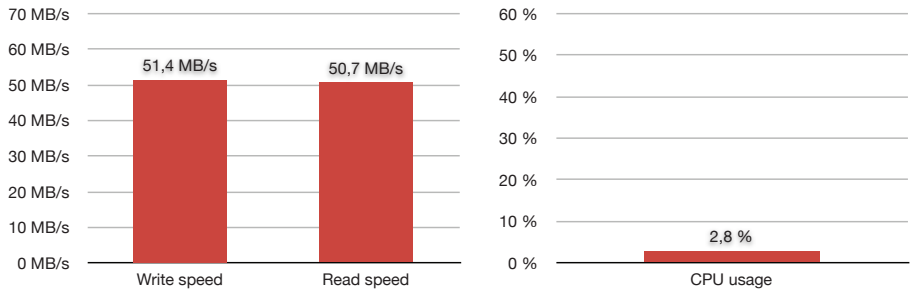



**Figure B.9:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows Vista without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 39.72 MB/s | 39.34 MB/s | 53.20 % |
| *#2* | 40.64 MB/s | 38.84 MB/s | 54.10 % |
| *#3* | 40.17 MB/s | 39.76 MB/s | 53.00 % |
| *#4* | 40.23 MB/s | 39.26 MB/s | 52.70 % |
| *#5* | 40.55 MB/s | 39.76 MB/s | 53.20 % |
| *#6* | 40.54 MB/s | 40.81 MB/s | 53.10 % |
| *#7* | 40.61 MB/s | 40.07 MB/s | 52.80 % |
| *#8* | 40.20 MB/s | 39.89 MB/s | 53.80 % |
| *#9* | 40.59 MB/s | 40.26 MB/s | 54.60 % |
| *#10* | 40.63 MB/s | 40.46 MB/s | 52.60 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **40.4 MB/s** | **39.8 MB/s** | **53.3 %** |
| *Standard deviation* | 0.30 | 0.59 | 0.65 |
| *Standard error* | 0.10 | 0.19 | 0.21 |
| *95% Confidence interval* | [40.17,40.60] | [39.43,40.26] | [52.84,53.76] |

**Table B.10:** *Disk performance measurements for computer 2 running Windows Vista with disk encryption.*

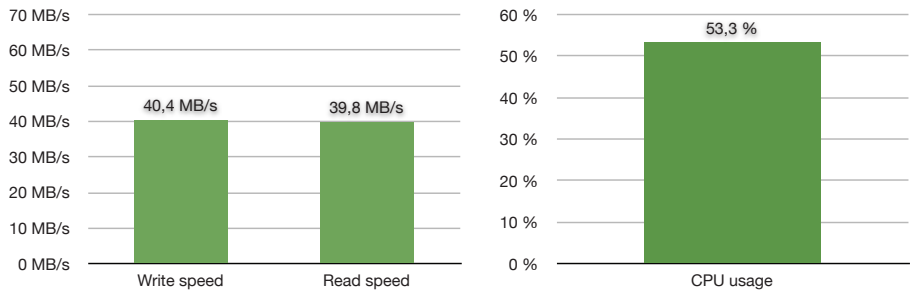



**Figure B.10:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows Vista with disk encryption.*

## B.6 Computer 2 using Windows 7

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 60.65 MB/s | 57.13 MB/s | 2.30 % |
| *#2* | 61.11 MB/s | 57.82 MB/s | 2.30 % |
| *#3* | 60.11 MB/s | 56.80 MB/s | 2.50 % |
| *#4* | 60.86 MB/s | 57.82 MB/s | 2.20 % |
| *#5* | 55.99 MB/s | 57.82 MB/s | 2.40 % |
| *#6* | 60.32 MB/s | 57.85 MB/s | 2.00 % |
| *#7* | 61.92 MB/s | 57.58 MB/s | 2.20 % |
| *#8* | 60.86 MB/s | 56.32 MB/s | 2.20 % |
| *#9* | 61.58 MB/s | 57.99 MB/s | 1.90 % |
| *#10* | 62.14 MB/s | 57.82 MB/s | 2.00 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **60.6 MB/s** | **57.5 MB/s** | **2.2 %** |
| *Standard deviation* | 1.73 | 0.56 | 0.19 |
| *Standard error* | 0.55 | 0.18 | 0.06 |
| *95% Confidence interval* | [59.33,61.77] | [57.10,57.89] | [2.07,2.33] |

**Table B.11:** *Disk performance measurements for computer 2 running Windows 7 without disk encryption.*

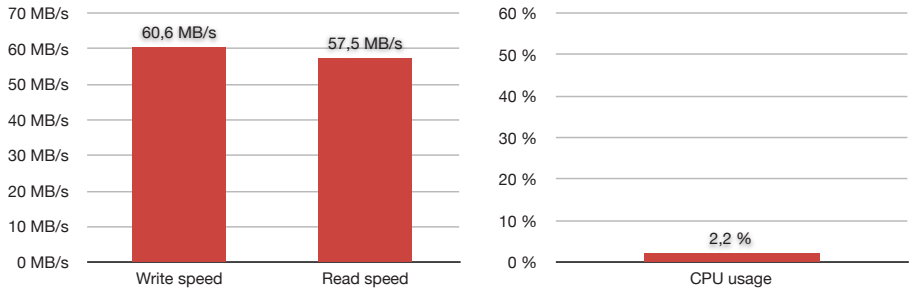



**Figure B.11:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows 7 without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 41.10 MB/s | 40.95 MB/s | 50.70 % |
| *#2* | 41.14 MB/s | 39.57 MB/s | 50.90 % |
| *#3* | 41.63 MB/s | 41.87 MB/s | 51.20 % |
| *#4* | 41.16 MB/s | 41.33 MB/s | 51.10 % |
| *#5* | 40.76 MB/s | 40.87 MB/s | 51.20 % |
| *#6* | 41.60 MB/s | 41.29 MB/s | 52.60 % |
| *#7* | 41.13 MB/s | 40.88 MB/s | 50.90 % |
| *#8* | 41.01 MB/s | 40.73 MB/s | 51.20 % |
| *#9* | 41.13 MB/s | 40.57 MB/s | 51.30 % |
| *#10* | 41.17 MB/s | 40.95 MB/s | 51.00 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **41.2 MB/s** | **40.9 MB/s** | **51.2 %** |
| *Standard deviation* | 0.26 | 0.60 | 0.52 |
| *Standard error* | 0.08 | 0.19 | 0.16 |
| *95% Confidence interval* | [41.00,41.36] | [40.48,41.32] | [50.83,51.57] |

**Table B.12:** *Disk performance measurements for computer 2 running Windows 7 with disk encryption.*

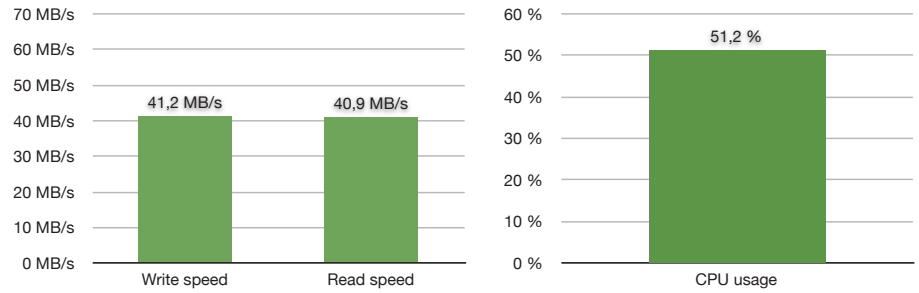



**Figure B.12:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 2 running Windows 7 with disk encryption.*

## B.7 Computer 3 using Windows XP

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 94.06 MB/s | 93.76 MB/s | 2.00 % |
| *#2* | 94.06 MB/s | 93.33 MB/s | 1.90 % |
| *#3* | 93.25 MB/s | 93.76 MB/s | 1.90 % |
| *#4* | 92.77 MB/s | 92.90 MB/s | 1.90 % |
| *#5* | 93.21 MB/s | 95.08 MB/s | 1.90 % |
| *#6* | 94.50 MB/s | 93.76 MB/s | 1.90 % |
| *#7* | 93.21 MB/s | 93.76 MB/s | 2.00 % |
| *#8* | 94.06 MB/s | 94.19 MB/s | 2.10 % |
| *#9* | 94.96 MB/s | 93.33 MB/s | 2.00 % |
| *#10* | 94.50 MB/s | 95.52 MB/s | 1.90 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **93.9 MB/s** | **93.9 MB/s** | **2.0 %** |
| *Standard deviation* | 0.71 | 0.80 | 0.07 |
| *Standard error* | 0.22 | 0.25 | 0.02 |
| *95% Confidence interval* | [93.36,94.36] | [93.37,94.51] | [1.95,2.05] |

**Table B.13:** *Disk performance measurements for computer 3 running Windows XP without disk encryption.*

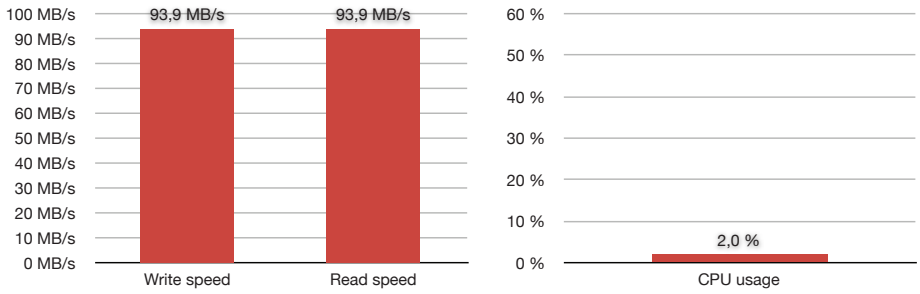



**Figure B.13:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows XP without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 63.03 MB/s | 66.61 MB/s | 45.10 % |
| *#2* | 62.89 MB/s | 68.64 MB/s | 45.60 % |
| *#3* | 64.12 MB/s | 63.66 MB/s | 45.80 % |
| *#4* | 67.54 MB/s | 65.69 MB/s | 46.60 % |
| *#5* | 61.40 MB/s | 67.70 MB/s | 45.30 % |
| *#6* | 63.58 MB/s | 67.14 MB/s | 46.10 % |
| *#7* | 59.68 MB/s | 61.56 MB/s | 45.70 % |
| *#8* | 64.12 MB/s | 67.23 MB/s | 46.10 % |
| *#9* | 61.47 MB/s | 67.54 MB/s | 45.40 % |
| *#10* | 62.46 MB/s | 68.01 MB/s | 46.00 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **63.0 MB/s** | **66.4 MB/s** | **45.8 %** |
| *Standard deviation* | 2.10 | 2.19 | 0.45 |
| *Standard error* | 0.66 | 0.69 | 0.14 |
| *95% Confidence interval* | [61.55,64.51] | [64.83,67.92] | [45.48,46.12] |

**Table B.14:** *Disk performance measurements for computer 3 running Windows XP with disk encryption.*

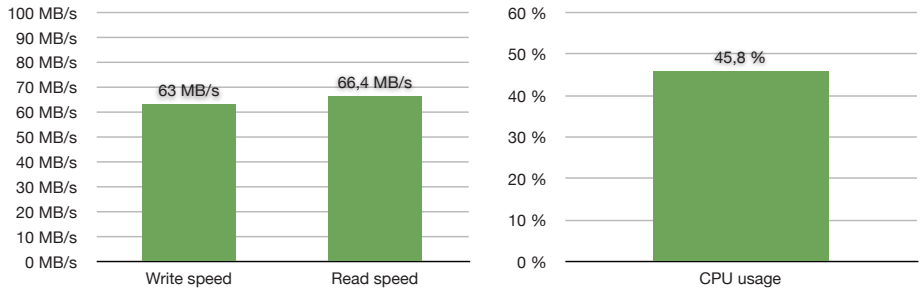



**Figure B.14:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows XP with disk encryption.*

## B.8 Computer 3 using Windows Vista

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 83.78 MB/s | 83.80 MB/s | 2.50 % |
| *#2* | 88.22 MB/s | 89.38 MB/s | 2.50 % |
| *#3* | 87.09 MB/s | 89.01 MB/s | 2.40 % |
| *#4* | 88.96 MB/s | 86.79 MB/s | 2.60 % |
| *#5* | 87.45 MB/s | 89.38 MB/s | 2.70 % |
| *#6* | 87.09 MB/s | 89.75 MB/s | 2.60 % |
| *#7* | 87.45 MB/s | 89.38 MB/s | 2.40 % |
| *#8* | 87.45 MB/s | 89.01 MB/s | 2.50 % |
| *#9* | 86.68 MB/s | 89.38 MB/s | 2.40 % |
| *#10* | 87.11 MB/s | 88.28 MB/s | 2.40 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **87.1 MB/s** | **88.4 MB/s** | **2.5 %** |
| *Standard deviation* | 1.34 | 1.83 | 0.11 |
| *Standard error* | 0.43 | 0.58 | 0.03 |
| *95% Confidence interval* | [86.18,88.08] | [87.13,89.71] | [2.42,2.58] |

**Table B.15:** *Disk performance measurements for computer 3 running Windows Vista without disk encryption.*

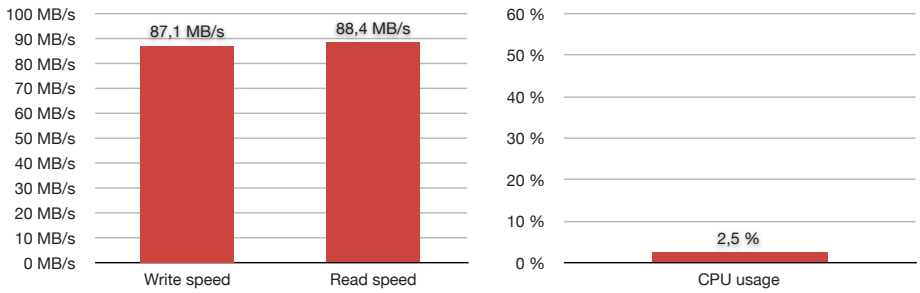



**Figure B.15:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows Vista without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 64.75 MB/s | 72.07 MB/s | 48.90 % |
| *#2* | 64.85 MB/s | 67.16 MB/s | 49.50 % |
| *#3* | 65.63 MB/s | 69.41 MB/s | 50.10 % |
| *#4* | 61.39 MB/s | 68.01 MB/s | 50.70 % |
| *#5* | 68.04 MB/s | 67.02 MB/s | 50.90 % |
| *#6* | 64.30 MB/s | 69.27 MB/s | 49.20 % |
| *#7* | 64.57 MB/s | 68.65 MB/s | 50.70 % |
| *#8* | 64.84 MB/s | 71.17 MB/s | 51.00 % |
| *#9* | 68.30 MB/s | 72.54 MB/s | 51.50 % |
| *#10* | 63.94 MB/s | 70.55 MB/s | 50.90 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **65.1 MB/s** | **69.6 MB/s** | **50.3 %** |
| *Standard deviation* | 1.98 | 1.95 | 0.87 |
| *Standard error* | 0.63 | 0.62 | 0.28 |
| *95% Confidence interval* | [63.66,66.46] | [68.21,70.96] | [49.69,50.91] |

**Table B.16:** *Disk performance measurements for computer 3 running Windows Vista with disk encryption.*

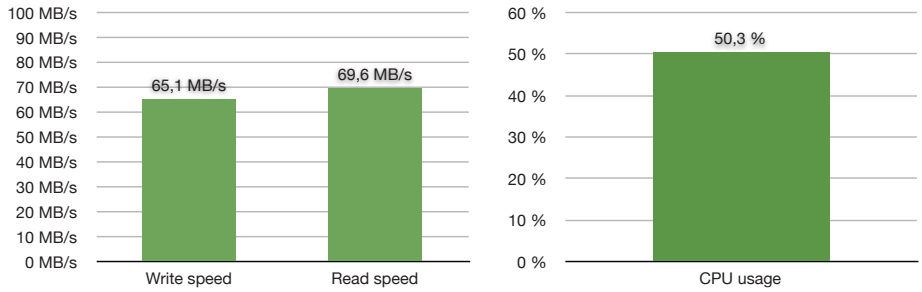



**Figure B.16:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows Vista with disk encryption.*

## B.9 Computer 3 using Windows 7

### Without disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 98.86 MB/s | 97.69 MB/s | 2.10 % |
| *#2* | 99.05 MB/s | 99.93 MB/s | 2.50 % |
| *#3* | 95.87 MB/s | 99.93 MB/s | 2.30 % |
| *#4* | 96.93 MB/s | 99.18 MB/s | 2.30 % |
| *#5* | 99.05 MB/s | 91.24 MB/s | 2.20 % |
| *#6* | 99.48 MB/s | 99.18 MB/s | 2.30 % |
| *#7* | 99.81 MB/s | 100.36 MB/s | 2.40 % |
| *#8* | 98.21 MB/s | 99.93 MB/s | 2.40 % |
| *#9* | 99.05 MB/s | 98.32 MB/s | 2.40 % |
| *#10* | 100.01 MB/s | 99.93 MB/s | 2.20 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **98.6 MB/s** | **98.6 MB/s** | **2.3 %** |
| *Standard deviation* | 1.30 | 2.71 | 0.12 |
| *Standard error* | 0.41 | 0.86 | 0.04 |
| *95% Confidence interval* | [97.72,99.55] | [96.66,100.48] | [2.22,2.39] |

**Table B.17:** *Disk performance measurements for computer 3 running Windows 7 without disk encryption.*

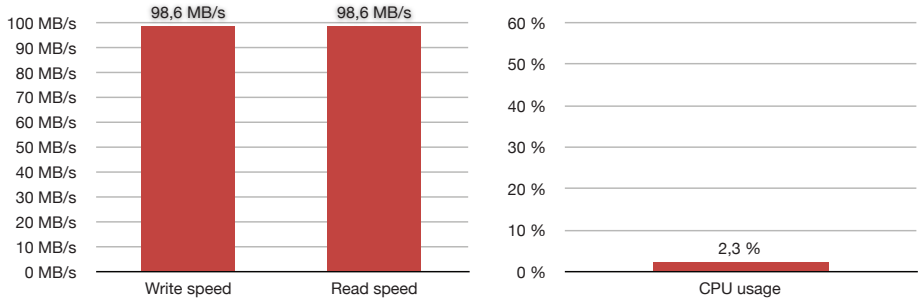



**Figure B.17:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows 7 without disk encryption.*

## With disk encryption

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *#1* | 54.59 MB/s | 62.39 MB/s | 53.70 % |
| *#2* | 55.81 MB/s | 63.37 MB/s | 52.80 % |
| *#3* | 54.36 MB/s | 62.87 MB/s | 54.20 % |
| *#4* | 54.46 MB/s | 64.07 MB/s | 52.90 % |
| *#5* | 54.88 MB/s | 63.73 MB/s | 53.20 % |
| *#6* | 52.75 MB/s | 63.80 MB/s | 53.60 % |
| *#7* | 54.31 MB/s | 63.80 MB/s | 53.50 % |
| *#8* | 54.99 MB/s | 56.18 MB/s | 54.00 % |
| *#9* | 53.14 MB/s | 63.74 MB/s | 53.20 % |
| *#10* | 54.46 MB/s | 63.81 MB/s | 53.60 % |

| | *Write speed* | *Read speed* | *CPU usage* |
|---|---|---|---|
| *Arithmetic average* | **54.4 MB/s** | **62.8 MB/s** | **53.5 %** |
| *Standard deviation* | 0.88 | 2.37 | 0.45 |
| *Standard error* | 0.28 | 0.75 | 0.14 |
| *95% Confidence interval* | [53.76,54.99] | [61.10,64.45] | [53.18,53.82] |

**Table B.18:** *Disk performance measurements for computer 3 running Windows 7 with disk encryption.*

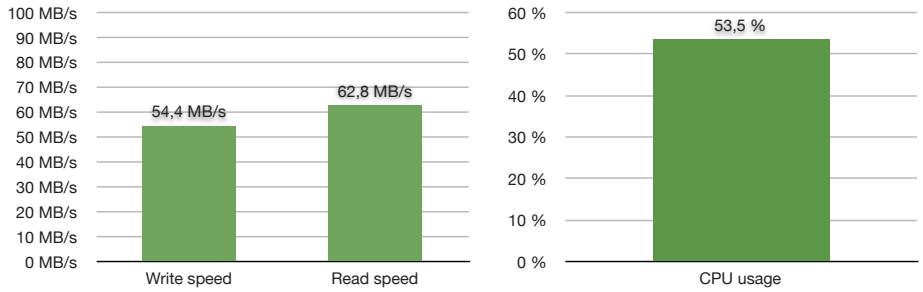



**Figure B.18:** *Average write speed, read speed, and CPU usage during disk benchmark for computer 3 running Windows 7 with disk encryption.*

# Appendix C

# Attached ZIP file

The attached ZIP file contains the following files and directories:

- `TrueCrypt6.1a.exe` – TrueCrypt 6.1a for Windows
- `HDTunePro3.5.exe` – HD Tune Pro 3.5 for Windows
- `Logs/Computer1/` – Log files for computer 1
- `Logs/Computer2/` – Log files for computer 2
- `Logs/Computer3/` – Log files for computer 3
- `Logs_README.txt` – Describing the contents of the `Logs`-folder