NTNU
Norwegian University of
Science and Technology

Study of the IEEE Standard 1619.1: Authenticated Encryption with Length Expansion for Storage Devices

Ignacio Gonzalez Torrego

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

The thesis will investigate the new IEEE P1619.1 standard for encryption and authentication algorithms suitable for data storage devices that support expanding blocks. All of these modes are using the NIST-approved AES-256 block cipher. The approved modes are as follows:

CCM-128-AES-256: Counter mode encryption with cipher block chaining message authentication code.
GCM-128-AES-256: Galois/Counter mode (counter mode encryption with 128-bit finite field message authentication code)
CBC-AES-256-HMAC-SHA: Cipher block chaining mode for encryption with keyhash message authentication code using secure hashing algorithm.
XTS-AES-256-HMAC-SHA: XTS encryption with key-hash message authentication code using secure hashing algorithm.

Assignment given: 03. February 2009
Supervisor: Danilo Gligoroski, ITEM

Para Ana, por su apoyo y cariño

Abstract

Ignacio González Torrego - iggontor AT gmail DOT com
Student at the Norwegian University of Science and Technology (NTNU)
Spring 2009

This Thesis will analyze the standard 1619.1 published by the IEEE. The aim of this standard is to provide authenticated encryption to stored data with AES algorithm working in XTS mode. XTS-AES is a 128-bit block cipher characterized by the use of two AES encryptions with two different keys of the same size, tweak values to add uncertainty to cipher data, (2^{128}) Galois fields and The Ciphertext Stealing technique for data units not perfectly divisible into 128-bit blocks. There is no unanimous agreement about the profits of this standard so various aspects such as the use of two different keys, implementation in other areas or the support of the storage industry will be a source of controversy. Some commercial software and hardware that implement XTS-AES encryption mode will be presented and used to test and analyze the security properties presented by the standard IEEE 1619.1.

Keywords

AES, XTS, Tweak values, Ciphertext Stealing, XTS-AES Comments

Preface

This Thesis is submitted to the Norwegian University of Science and Technology (NTNU).

This Thesis has been performed at the Department of Telematics, NTNU, Trondheim, with Danilo B. Gligoroski as main supervisor.

Acknowledgements

I would like to express my gratitude towards those who made this Thesis possible, those who supported me during the work as well as those with whom it has been a true pleasure to work.

I would like to express my sincere thanks to Prof Danilo B. Gligoroski from the Norwegian University of Science and Technology (NTNU), an admirable professor, a really hard worker and an expert in the field of Information Security for his great help. His support and encouragement, together with the interesting and helpful discussions, are sincerely acknowledged.

And last but by no means least, I would like to express my gratitude towards my family and Ana for their moral support.

Ignacio González Torrego

Trondheim, June 2009

Contents

List of Tables

List of Figures

Abbreviations

AES	Advanced Encryption Standard
AES-CCM	AES with Counter with CBC-MAC
AES-GCM	AES with Galois/Counter Mode
ASIC	Application-Specific Integrated Circuit
CBC	Cipher-Block Chaining
CFB	Cipher FeedBack
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic CodeBook
ECC	Elliptic Curve Cryptography
FAQ	Frequently Asked Questions
FAT	File Allocation Table
FIPS	Federal Information Processing Standards
FPGA	Field Programmable Gate Array
GF	Galois field
IEEE	The Institute of Electrical and Electronics Engineers
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NSA/CSS	National Security Agency/Central Security Service
NTFS	New Technology File System
OCB	Offset CodeBook
OFB	Output FeedBack

PRP	PseudoRandom Permutation
QKD	Quantum Key Distribution
SISWG	Security in Storage Working Group
TAE	Tweakable Authenticated Encryption
TBC	Tweak Block Chaining
TBC	Tweakable Block Cipher
TCB	Tweaked CodeBook
TCH	Tweak Chain Hash
TRD	TrueCrypt Rescue Disk
UAC	User Account Control
XCB	X protocol C-language Binding
XEX	Xor-Encrypt-Xor
XOR	Exclusive OR
XTS	XEX encryption mode with Tweak and ciphertext Stealing

Chapter 1

Introduction

1.1 Problem outline

Nowadays, security is a vital part in our way of life. Cars are safer and prevent possible injury to the occupants and people outside them, houses have complex alarms and powerful systems to prevent burglary, companies hire firms specializing in security to prevent thefts and assaults, etc.

For that reason, it is not a surprise that protecting information is a key objective at the moment, not only at government level (classified files, future projects, economic, military defense) but also business (patents, business strategies, development of new products), administrative (medical files, criminal records) and private (personal details, bank accounts, sensitive information). A way to hide sensitive information or manipulate it is needed so that even if an attacker finds it, it would be unable to realize what is really watching.

The risk of compromised data security is becoming ever greater. These risks can be classified into three classes:

- Unintentional errors of individuals and/or machines.
- Natural disasters.
- Intentional attacks.

The first being the most common, about 80% of cases.

The protection of information is more acute since the emergence of telematic networks. These networks and especially the Internet, make information a global problem and not isolated to machines within the company. Technologies applied to network security are in their initial development phase, for two reasons:

- Most operating systems are designed for mainframe/terminal architectures and not for client/server or Internet/Intranet architectures that are being used today.

- There are no standards or global organizations accepted by all companies specialized in providing security.

Cryptology was born long ago to protect information. Its primary intention is to ensure the transmission medium by which information will be sent to avoid as much as possible the access to any unauthorized person. Cryptography is a branch within the science of Cryptology, and is responsible for encrypting and decrypting information using special techniques.

1.2 Cryptography

Cryptography *(from Greek kryptós, "hidden, secret"; and grápho, "I write").*

Cryptography is the art, science or art of writing secret. The basic principle of cryptography is to maintain the privacy of communication between two people by altering the original message so that it is incomprehensible to anyone other than the addressee, to which we must use authentication, i.e. the signature of the message so that a third person can not impersonate the sender. A transformation of the original message in the encrypted message (cryptogram) is called encryption, and the reverse is called decryption, these steps are executed using a predefined set of rules among sender and receiver and that we call it "key". Cryptanalysis is the set of techniques that try to find the key used between sender and receiver, thus revealing the secret of his correspondence.

In the cryptography, it is possible to distinguish between classical cryptography, which includes all references cryptographic before 1st century BC, and modern cryptography, from the 15th century.[1] Tables 1.1 and 1.2 show various landmarks in the history of cryptography.

Classical Cryptography	Medieval Cryptography
Scytale (V century BC) 1.1	Alberti Cipher Disc (XV century) 1.2
Polybios cipher (II century BC)	Jefferson disc (XVIII century)
Caesar cipher (I century BC)	Wheatstone disc (XIX century)

Table 1.1: Classical and medieval cryptography

What are the current trends with regard to modern cryptography? Elliptic curve cryptography (ECC) and Quantum key distribution (QKD) are the new cryptographic goals toward which all are directing efforts.

- In 1985 Neil Koblitz and Victor Miller proposed Elliptic Curve Cryptosystem (ECC), or elliptic curves cryptosystems whose security is based on the same problem that

[1]After the 1st century BC and until the 15th century AD, was not aware of any new invention of cryptographic system (at that time was known as the Dark Ages because there were more regression than progress in absolutely every facet of human knowledge and cryptography would not be an exception).

II Word War Cryptography	Modern Cryptography
Enigma 1.3	Blockciphers
Hagelin	Pseudorandom Functions
M-325	Symmetric Encryption
Vigenère cipher	Hash Functions
Beaufort cipher	Message Authentication
Playfair cipher	Digital Signatures
Hill cipher	Authenticated Encryption
	Computational Number Theory
	Number-Theoretic Primitives
	Asymmetric Encryption
	Digital signatures
	Authenticated Key Exchange

Table 1.2: II World War and Modern Cryptography [1]

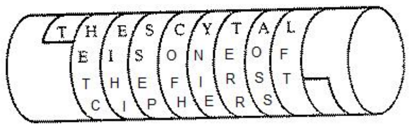


Figure 1.1: Scytale

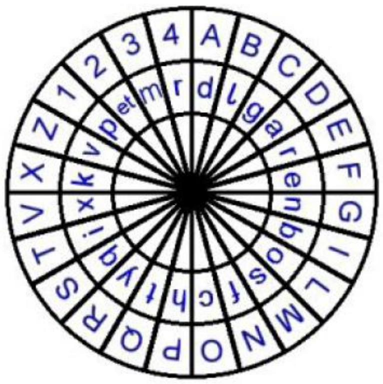


Figure 1.2: Alberti disc [2]

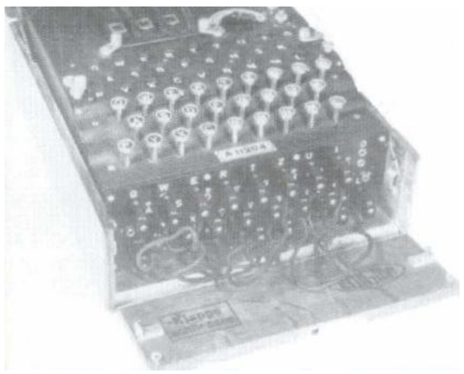


Figure 1.3: Enigma machine [3]

the methods of DSA and Diffie-Hellman, but instead of using numbers as symbols of the alphabet to encrypt the message, using points in a mathematical object called elliptic curves. ECC can be used both to encrypt and to digitally sign. Until now, any attack is not known whose execution time is expected sub exponentially to break the ECC, this makes that for obtaining the same level of security provided by other systems, the space key ECC is much smaller, what makes a technology appropriate for use in environments restricted resources (memory, cost, speed, bandwidth, etc.)

- Quantum key distribution method is based on the Heisenberg uncertainty principle, which states that simply by observing, changing what is being observed, i.e. is not possible to know two different properties of a subatomic particle in a single instant of time.

1.3 Block Ciphers

In cryptography, a cipher block is a symmetric key cipher which operates on groups of bits of fixed length, called blocks, applying an invariant[2] transformation. When performing encryption, a block cipher takes a block of plaintext or clear input and produces a block of ciphertext of equal size. This transformation takes place under the action of a user-provided secret key. Decryption is similar, but now the ciphertext block is the input and the output is the corresponding plaintext block. [4]

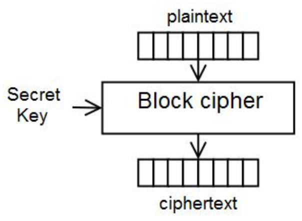


Figure 1.4: Block cipher encryption process

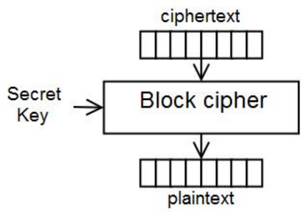


Figure 1.5: Block cipher decryption process

Block ciphers are different of stream ciphers because they transform the plaintext processing each bit individually, one after another, and the transformation varies during the encryption process. The difference between the two types of units is somewhat fuzzy, since a block cipher can be operated in a mode that allows use as a stream cipher unit, where instead of digit operating blocks.

The Data Encryption Standard (DES) was a design of block cipher of great influence. Was developed and published by IBM and released in 1977 as standard. The Advanced

[2]In mathematics, an entity is considered invariant under a transformation if the transformed image of the entity is indistinguishable from the original entity.

Encryption Standard (AES)[3] is a successor to DES, which was adopted in 2001.

To encrypt messages longer than the block size, different mode of operation are used:

- Electronic Code Book (ECB) is the easiest mode of operation. It consist on breaking the plaintext into blocks $P_1, P_2 ... P_n$ and then encrypt each of them independently.

  $$C_i = E_k(P_i)$$

  Its main weakness is if the same plaintext is encrypted twice or more times, the same ciphertext will be generated. The typical application for ECB is the secure transmission of short pieces of information, as for example temporary encryption keys.

- Cipher Block Chaining (CBC). As in ECB mode, the plaintext is divided into blocks $P_1, P_2 ... P_n$ but now the encryption of each block of plaintext depends on the previous ciphertext block: before encrypt a plaintext block, it is XORed with the previous ciphertext block. To encrypt the first block of plaintext, an Initial Vector (IV) is used.

  $$C_i = E_k(C_{i-1} \oplus P_i)$$
  $$C_0 = IV$$

  The major advantages of this mode is that the encryption of a block depends not only on the current but in all the previous blocks, so a repeated plaintext block will be encrypted differently.

- Cipher FeedBack (CFB) mode is different of the modes listed above because the Key is now different to each plaintext block. The first key is created from an Initial Vector, the following keys from the previous ciphertext block.

  $$C_i = P_i \oplus K_i$$

  Advantages: the block cipher is used as a stream cipher and is appropriate when data arrives in segments. Disadvantages: A corrupted ciphertext segment during the transmission will affect the current and several following plaintext segments when decryption is tried.

- Output FeedBack (OFB) mode is similar to CFB, but now the each key is generated from the previous one. With this variation is achieved the avoidance the propagation of a corrupted ciphertext segment to the following decrypted segments.

- The basic idea of Counter mode is create a key stream from a single key to encrypt the different plaintext blocks.

  $$T_1 = IV$$
  $$T_i = T_{i-1} + 1$$
  $$C_i = P_i \oplus E_K(T_i)$$

[3]We will deeply analyze AES in 2.2

$$C = (IV, C_1, C_2, ...)$$

The main advantages of this mode of operation are: fast encryption and decryption because blocks can be processed in parallel, IV should not be used several times.

1.4 IEEE

The IEEE name was originally an acronym for the Institute of Electrical and Electronics Engineers, Inc. Nowadays, the field of study of this organism is so extend that they are not only focused on those fields so they prefer to be named using their initials: IEEE.

Its creation dates back to 1884, counting among its founding to personalities such as Thomas Alva Edison, Alexander Graham Bell and Franklin Leonard Pope. In 1963 they adopted the name IEEE of the merge of the associations AIEE (American Institute of Electrical Engineers) and IRE (Institute of Radio Engineers).

Through its members, over 360,000 volunteers in 175 countries, IEEE is a leading authority and most prestigious in different technical areas such computer engineering, biomedical and aerospace technologies, areas of electricity, control, telecommunications and consumer electronics, among others.

According to the IEEE, its job is to promote creativity, development and integration, sharing and applying advances in information technology, electronics and general science for the benefit of humanity and the professionals themselves.

1.5 Thesis Structure

Throughout this chapter we have made an overview of the current state of the cryptographic field and their characteristics as we presented SISWG belonging to the IEEE and have been the authors of the XTS-AES encryption that we will develop during the following chapters:

- Chapter 2: Background
- Chapter 3: IEEE 1619.1
- Chapter 4: Tools: Software and Hardware
- Chapter 5: Methods and results
- Chapter 6: Conclusion

In the second chapter, named Background (2), we will presents to the reader the background needed to deeply understand the theory in which is based the standard IEEE 1619.1 described in (3). Chapter 4 will introduce the software (4.1) and the hardware (4.2) used to test the functionality of the XTS-AES encryption mode under study, the results will be shown in (5) and we will analyze it comparing the different software encryption tools. Finally conclusion and a future work will be presented in the last chapter (6).

Chapter 2

Background

This section of the Thesis is intended to provide the reader a comprehensive idea about the current situation in which the cryptographic sector is present as well as analyze the several innovations that incorporate the standard XTS-AES.

First we will see an overview of the characteristic of the standard under review and a description of the different components which allow encryption of stored data as: tweak vectors, Galois fields, AES encryption and XTS variation of the AES standard.

In the following topic we pretend to show the characteristics of the AES encryption algorithm, which is the basis of the standard under study. Analyzing how AES uses the key entered for building with it the separate subkeys for each of the rounds, and how AES is able to manipulate the text entered. To do this we will see, step by step, which are the changes made on the plaintext to provide uncertainty to the ciphertext.

After knowing how AES works, we must know what is the XTS mode of operation and how can increase the security of the AES algorithm. Also we want to analyze why it has been introduced in the IEEE standard and has been gradually included in the encryption software and the new IP cores.

Then we will enter in the world of Galois fields (also called finite fields). We will see what are they and what its main features are. In the next chapter we will see how Galois fields are useful in the new encryption standard.

Finally we will analyze the last of the features of the standard that provide more security for stored data. We refer to tweak values. Specifically tweak values will introduce randomness to the ciphertext, thereby will prevent the encryption system to be deterministic and will add uncertainty to a potential attacker.

2.1 Overview

The security of data stored in a digital format has grown in importance and that is why it is usual for large amounts of digital information, secret and extremely sensitive to their owners, to be entrusted to organizations specifically dedicated to this task, or outside recorded in a digital format and subsequently saved physically locked to prevent theft and the subsequent acquisition of data and confidential information.

For this reason, the IEEE Computer Society established the Security in Storage Working Group (SISWG) in 2002 along with the 1619 draft as a way to standardize the way in which data is safe. The objective is to build a standard architecture that could protect the data from the time they are created until a duly authorized person wishes to access them, without taking into account the way in which data is transmitted or stored.

The IEEE 1619 standard covers a wide range of topics that include:

- Tape Encryption
- Wide Encryption
- Key Management

This Thesis will focus on the first topic, Tape Encryption, which is deeply developed in the draft 1619.1 Standard for Authenticated Encryption with Length Expansion for Storage Devices.

In the next sections of this chapter, we will introduce to the reader the basic concepts to undestand how works XTS-AES and why is a good way for storing data securely.

2.2 AES

Advanced Encryption Standard (AES), also known as Rijndael[1], was published on November 26, 2001 by the National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) after 5 years of a normalization process in which fifteen competing designs were submitted and evaluated before Rijndael was selected as the most appropriate.[5]

This section shows the reader how the Rijndael algorithm works, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths.

The algorithm specified here will be referred to as ”the AES algorithm”. The algorithm may be used with the three different key lengths indicated above, and therefore these different ways of work may be referred to as ”AES-128”, ”AES-192”, and ”AES-256”.[6]

[1]Rijndael is the name of the original algorithm in which its two Belgian developers, Joan Daemen and Vincent Rijmen, were presented to the AES selection process

We will show the reader the main features of AES algorithm that will be helpful to understand the basic operation of the standard at issue:

- Notation and conventions used in the algorithm specification, including the ordering and numbering of bits, bytes, and words;
- Mathematical properties that are useful in understanding the algorithm;
- Algorithm specification, covering the key expansion, encryption, and decryption routines;
- Implementation issues, such as key length support, keying restrictions, and additional block/key/round sizes.

AES is based on a design principle named Substitution permutation network and it presents some advantages: it is fast in both software and hardware, is relatively easy to implement and not require a big amount of memory. One of the difference between AES and its predecessor DES, is that AES do not use a Feistel network.

AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, in difference with Rijndael that was developed to work with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

Assuming one byte equals 8 bits, the fixed block size of 128 bits is 128 œ 8 = 16 bytes. AES operates on a 4Œ4 array of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field[2].

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plain-text into the final output of cipher-text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher-text back into the original plain-text using the same encryption key. [5]

Now, this section pretends to introduce to the reader the different stages that make up each of the iterations of the Rijndael algorithm:

- Key Expansion
- Initial Round
  - AddRoundKey
- Rounds
  - SubBytes
  - ShiftRows
  - MixColumns

[2]Arithmetic in a finite field is different from standard integer arithmetic. There are a limited number of elements in the finite field; all operations performed in the finite field result in an element within that field.

  - AddRoundKey
- Final Round
  - SubByte
  - ShiftRows
  - AddRoundKey

2.2.1 Key Expansion

The AES algorithm takes the Cipher Key, K, and begin the Expansion Key routine to generate all the subkeys that are used in different rounds.

The Key Expansion generates a total of Nb (Nr + 1) words: the algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data.

- *Subword()* $\rightarrow$ takes a four-byte input word and applies the S-box to the each of the four bytes to produce an output word
- *RotWord()* $\rightarrow$ takes a word $[\begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \end{array}]$ as input, then performs a cyclic permutation and returns the word $[\begin{array}{cccc} a_1 & a_2 & a_3 & a_0 \end{array}]$
- *Rcon[i]* $\rightarrow$ is the round constant word array and contains the values given by the array $[\begin{array}{cccc} x^{i-1} & 00 & 00 & 00 \end{array}]$, where x^{i-1} are the powers of x

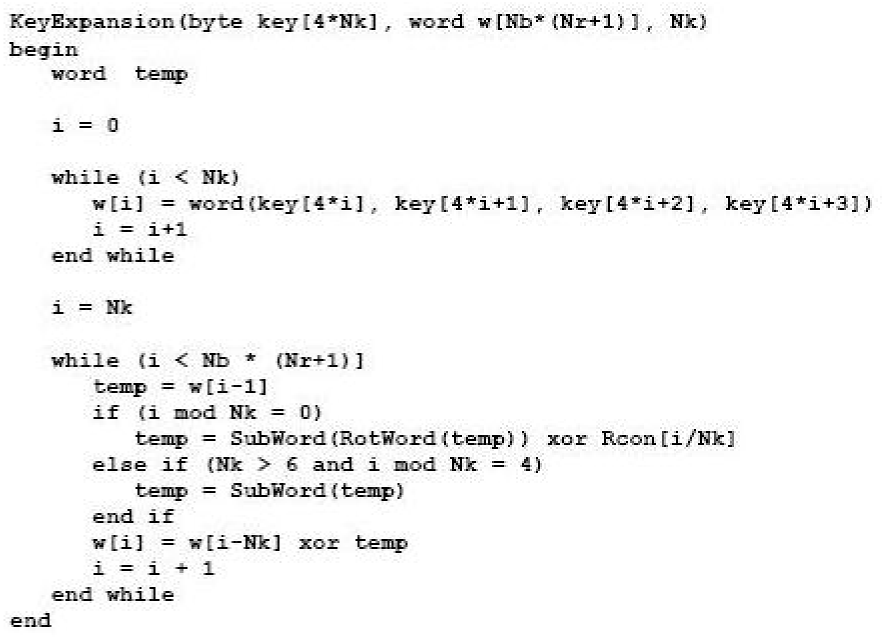
At the beginning, the first Nk words of the Expanded Key are filled with the Cipher Key. After that as we can see at the code of figure 2.1, every following word, w[i], is equal to the XOR of the previous word, w[i-1], and the word Nk positions earlier, w[i-Nk]. For words in positions that are a multiple of Nk, a transformation is applied to w[i-1] prior to the XOR, followed by an XOR with a round constant, Rcon[i]. This transformation consists of a cyclic shift of the bytes in a word (RotWord()), followed by the application of a table lookup to all four bytes of the word (SubWord()).[6]

2.2.2 SubBytes

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State[3] using a substitution table (S-box).

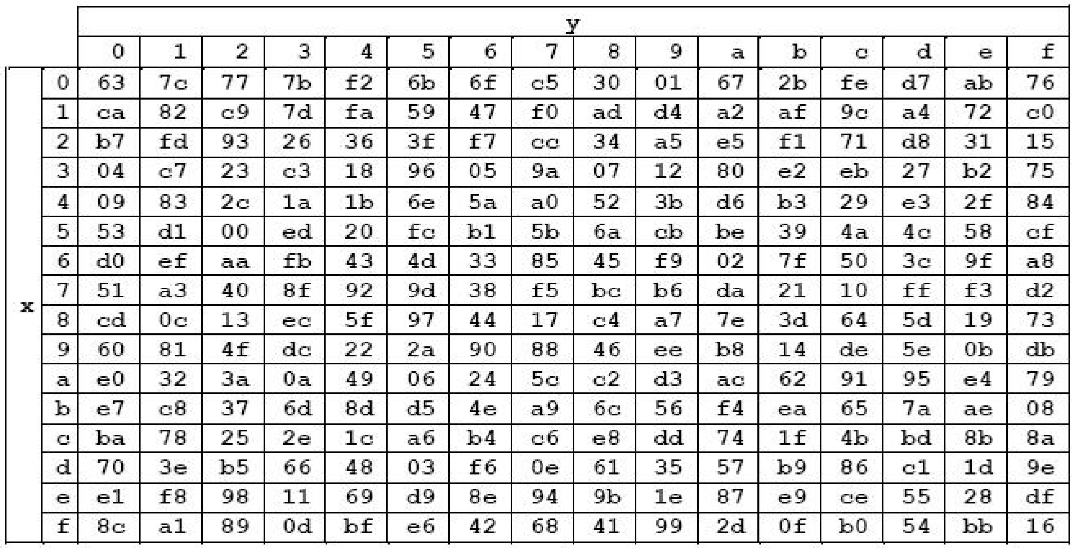
The S-box (see figure 2.2) is created to avoid attacks based on simple algebraic properties, is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any byte will be at the same position after the SubBytes step, and also to avoid any byte could be at its opposite position.

[3]Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes, each containing Nb bytes, where Nb is the block length divided by 32.[6]



```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Figure 2.1: Key Expansion Code[6]



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.2: S Box: Substitution values for the byte xy[6]

For example, if $a_{2,2} = (19)$, then the substitution value will be determined by the intersection of the row ”1” and the column ”9” in the S box(fig 2.2). This would result in $b_{2,2} = (d4)$.

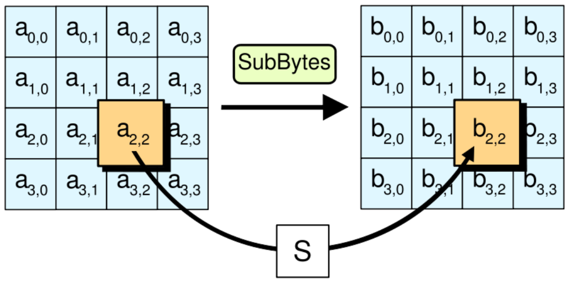


Figure 2.3: SubBytes Step[5]

2.2.3 ShiftRows

In the ShiftRows() step, the rows of the the State are rotated depending of its index. The first row. r=0 is not shifted. The bytes of the first row are shifted once, the bytes of the second row are shifted twice and the bytes for the third row are shifted three times.

Figure 2.4 illustrates the ShiftRows() transformation.

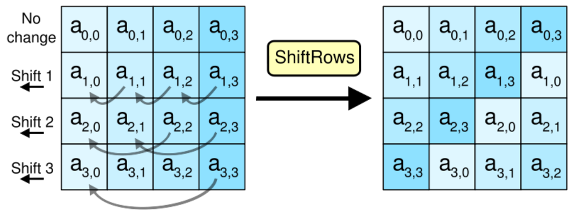


Figure 2.4: ShiftRows Step[5]

2.2.4 MixColumns

This step provides diffusion to the cipher. The MixColumns() transformation operates on the State column-by-column. Each column is treated as a four-term polynomial. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial c(x), given by $c(x) = 03x^3 + 01x^2 + 01x + 02$, as we can see in 2.5

2.2.5 AddRoundKey

In the AddRoundKey() step, a Round Key is finally added to the State by a simple bitwise XOR operation. Each Round Key is a subkey of Nb words from the key schedule

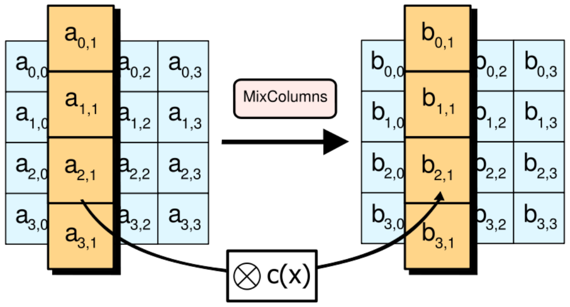


Figure 2.5: MixColumns Step[5]

(described in 2.2.1). Those Nb words are each added into the columns of the State as is shown in figure 2.6

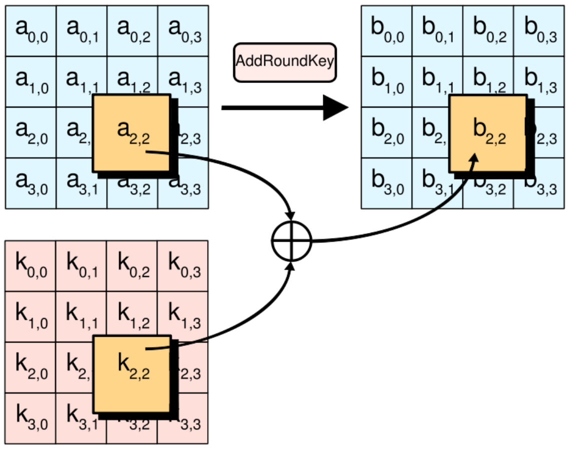


Figure 2.6: AddRoundKey Step[5]

2.3 XTS mode of operation

XTS is XEX-based Tweaked CodeBook mode (TCB) with Ciphertext stealing (CTS)[4]. Ciphertext stealing provides support for sectors with size not divisible by block size.[7]

XTS mode is, in fact, XEX mode designed by Phillip Rogaway in 2003, with a little modification: Instead XEX mode uses a single key for two different purposes, XTS mode uses two independent keys. XTS mode was approved as the IEEE 1619 standard for cryptographic protection of data on block-oriented storage devices in 2007.

For understand how XTS mode works, we pretend to show the reader an overview of XEX mode.

[4]XEX-TCB-CTS should be abbreviated as XTC but "C" was replaced with "S" (for "stealing") to avoid confusion with the abbreviated *[[Methylenedioxymethamphetamine/ecstasy]]*.

XEX was designed to be a strong tweakable block cipher. Is another tweakable encryption mode that execute a very good processing of consecutive blocks. In XEX mode, the entire key is divided in two parts of equal size: $K = K_1|K_2$, one of them is used to encryt the tweak value and combine the result with the plaintext of the block that will be encrypted with the second part of the key.

For example, we pretend to encrypt block j in sector S, the following formula is used and represent the working mode of XEX mode: $C = E_{K_1}(P \oplus X) \oplus X$ where $X = E_{K_2}(S) \oplus \alpha^j$ and α is the primitive element of $GF(2^{128})$ defined by polynomial x.(See 2.4)

As we will see in 2.5, tweak values are built by the combination of the sector address and the index of the block inside the sector where is stored. Tweak values produce variability and provide some randomly to the cipher text.

The next thing we need to know to completely understand XTS mode is the term Cipher-Text Stealing. CipherText Stealing refers to the messages that are not evenly divisible into blocks without resulting in any expansion of the ciphertext.

Is a technique of altering processing of the last two blocks of plaintext, resulting in a reordered transmission of the last two blocks of ciphertext and no ciphertext expansion is produced. This is accomplished by padding the last block (which is possibly incomplete) with the high order bits from the second to last ciphertext block (stealing the ciphertext from the second to last block). The (now full) last block is encrypted, and then exchanged with the second to last ciphertext block, which is then truncated to the length of the final plaintext block, removing the bits that were stolen, resulting in ciphertext of the same length as the original message size.[8]

2.4 Galois Fields

Galois Fields (GF) get their name from Évariste Galois and are also called finite fields. A GF is a field that has a finite field order[5] and that order is always a prime number or a power of a prime number. For each prime power there exists exactly one finite field $GF(p^n)$.

$GF(p)$ is called the prime field of order p, and is the field of residue classes modulo p, where the p elements are denoted $0, 1, ..., p-1$. $a = b$ in $GF(p)$ means the same as $a \equiv b\,(mod\,p)$.[9]

for example, the elments of the finite field GF(2) ara 0 and 1 and they satisfy both addition and multiplication tables shown in tables 2.1 and 2.2

A polynomial over GF(2) is irreducible if it cannot be factored into non-trivial polynomials. For example, x^2+x+1 is irreducible, but x^2+1 is not, since $x^2+1 = (x+1)(x+1)$.

[5]Field order refers to the number of elements of the field.

+	0	1
0	0	1
1	1	0

Table 2.1: GF(2) addition table

x	0	1
0	0	0
1	0	1

Table 2.2: GF(2) multiplication table

A polynomial over GF(2) is primitive if it has order $2^n - 1$. For example, $x^2 + x + 1$ has order $3 = 2^2 - 1$ since $(x^2 + x + 1)(x + 1) = x^3 + 1$. Thus $x^2 + x + 1$ is primitive. And let α denote one of the roots of a primitive polynomial, so a polynomial $F(x)$ with coeficients in $GF(p) = Z/pZ$ is a primitive polynomial if it has a root α in $GF(p^m)$ such that $\{0, 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{p^m-2}\}$[6] is the entire field $GF(p^m)$, and also, $F(x)$ is the smallest degree polynomial having α as a root.

For the rest of the Thesis, α^j would be considered as one of the root of the polynomial $F(x)$.

2.5 Tweakable Block Cyphers

As the authors[7] said in the presentation paper, TBC wants to became in a new cryptographic primitive and add a new input to the cipher: ”the tweak”. As we will see during this section, the tweak acts like an Initialization Vector in CBC mode or a Nonce in OCB mode. But, what are the advantages to add a new input to the cipher and what is ”the tweak”? The main features are named in [10]:

1. TBC are easy to design.
2. The extra cost of making a block cipher ”tweakable” is small.
3. It is easier to design and prove modes of operation based on TBC.

The ciphertext $C \in \{0,1\}^n$ in a conventional block cipher, is the result of the encryption of the message $M \in \{0,1\}^k$ with the key $K \in \{0,1\}^n$, as we can see in figure2.7.

Modes of operation began to use for encrypt messages with arbitrary length. They take the Key $K \in \{0,1\}^n$, an initialization vector (or nounce) $V \in \{0,1\}^v$ and a message $M \in \{0,1\}^*$ as input and produce a ciphertext $C \in \{0,1\}^*$ also with arbitrary length.

[6]If α in $GF(p^m)$ is a root of a primitive polynomial $F(x)$ then since the order of α is $p^m - 1$ that means that all elements of $GF(p^m)$ can be represented as successive powers of α

[7]Moses Liskov, Ronald L. Rivest and David Wagner

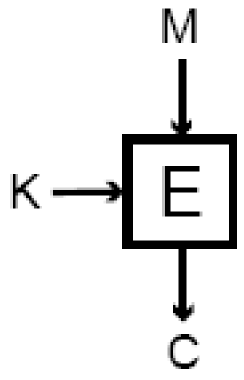


Figure 2.7: Block Cipher Encription[10]

Block ciphers have a serious problem that greatly compromises the security and privacy of the ciphertext and is that they are inherently deterministic, i.e. several encryptions of the same message with the same key will always produce the same ciphertext with corresponding weaknesses in security that this implies. To try to resolve that problem a conflict started about what it was better, keep the same key to enforce the efficiency or introduce variability in the encryption. This conflict usually end with the idea of using the same key, but adding variability before encrypting, after or both.

And that is the main idea of TBC, that maintain the same Key during the encryption process and also adds a new input, the tweak $T \in \{0,1\}^t$ that provides variability, to the message $M \in \{0,1\}^n$ and the $K \in \{0,1\}^k$ to produce the ciphertect $C \in \{0,1\}^n$. Those tweak values act like an initialization vectors or nounces.

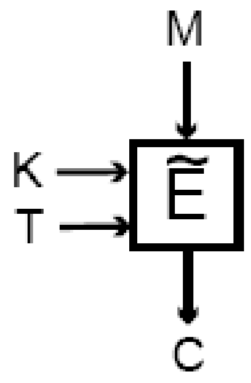


Figure 2.8: Tweak Block Cipher Encryption[10]

There are some objectives that TBC may accomplish to be as efficient as possible. TBC should have the property that changing the tweak should be easier that changing the key.

TBC should be secure also, even if an intruder takes control of the tweak input, TBC should remain secure: each fixed setting of the tweak gives rise to a different, apparently independent, family of standard block cipher encryption operators[10]. The main goal of TBC is that should be efficient: Both encryption and decryption should be easy to compute

It is necessary to distinguish the roles of the key and the tweak. On one hand, the key must be used to introduce uncertainty to the ciphertext, on the other hand the tweak should introduce variability but is not intended to introduce more uncertainty to the ciphertext.

Like block ciphers, TBC also allows variations the mode of operation due to the new input. Those modes are:

- Tweak Block Chaining (TBC)
- Tweak Chain Hash (TCH)
- Tweakable Authenticated Encryption (TAE)

2.5.1 Tweak Block Chaining

TBC mode of operation is similar to cipher block chaining (CBC). Instead of using an initialization vector, an initial tweak T_0 is used. Each successive message block M_i is encrypted using the key K and the tweak T_{i-1} that is the ciphertext for the previous block: $T_i = C_i$.

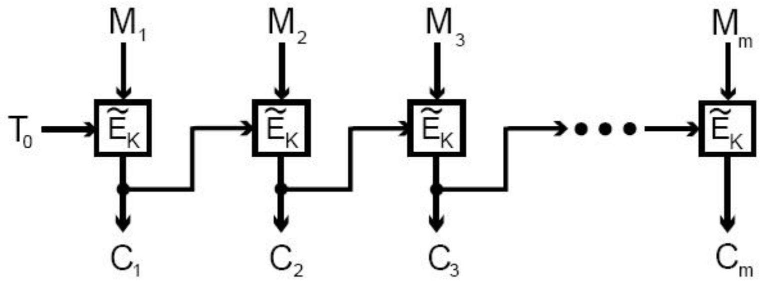


Figure 2.9: Tweak Block Chaining[10]

It is also possible to encrypt messages whose length is not a multiple of n, as it is shown in figure 2.10 a variant of the ciphertext-stealing is used.

2.5.2 Tweak Chain Hash

One possibility to make a hash function, is to adapt the Matyas-Meyer-Oseas construction to our TBC. We can use T_0 as the first tweak value and then perform the new tweak value

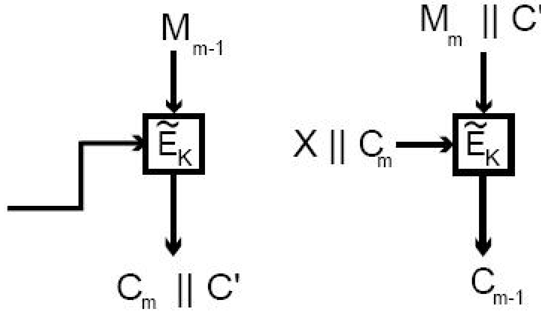


Figure 2.10: TBC ciphertext-stealing. Let r denote the length of the last (short) block M_m of the message. Then $|C_m| = |M_m| = r$ and $|C'| = n - r$. Here X denotes the rightmost $n - r$ bits of C_{m-2}[10].

for next block as the or-exclusive addition with the message and the ciphertext. Figure 2.11 shows the procedure of this operation mode.

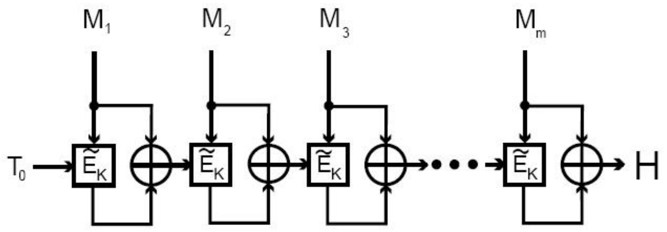


Figure 2.11: Tweak Chain Hash. The value of H is the output of the hash function. [10]

2.5.3 Tweakable Authenticated Encryption

The last mode of operation, TAE, is similar to OCB described in [11]. It could be viewed a rewrite of OCB using TBC instead DESX-like modules.

This mode takes as input an $(n/2)$-bit nonce N. The tweak Z_i for $i > 0$ is defined as the concatenation of the nonce N, an $(n/2 - 1)$-bit representation of the integer i, and a zero bit 0: $Z_i = N||i||0$. The tweak Z_0 is defined as the concatentation of the nonce N, an $(n/2 - 1)$-bit representation of the integer b, where b is the bit-length of the message M, and a one bit 1: $Z_0 = N||b||1$. The message M is divided into $m-1$ blocks $M_1, ..., M_{m-1}$ of length n and one last block M_m of length r for $0 < r \leq n$ (except that if $|M| = 0$ then the last (and only) block has length 0). Each ciphertext block C_i has same length as M_i. The function $len(M_m)$ produces an (n)-bit binary representation of the length r of the last message block. The last message block M_m is padded with zeros if necessary to make it length n before xoring. The checksum is $(M_1 \oplus ... \oplus M_{m-1} \oplus (M_m||0^*))$. The parameter τ, $0 \leq \tau \leq n$ specifies the desired length of the authentication tag.[10]

As OCB mode proposed by Rogaway, TAE achieve all the security features that characterize the OCB mode:

- Unforgeable
- Pseudorandom

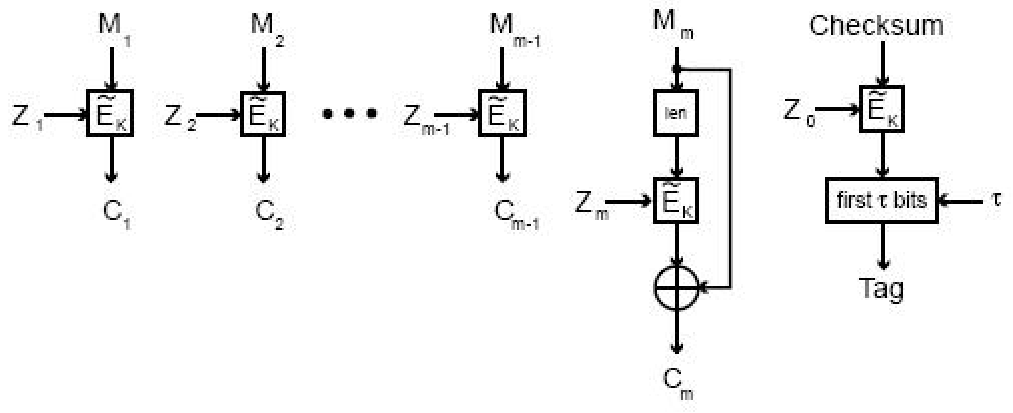


Figure 2.12: Tweakable Authenticated Encryption [10]

Chapter 3

IEEE 1619.1

In this chapter we will focus on the main issue and which is the principal subject of this Thesis: *IEEE Std 1619, IEEE Standard for Standard for Authenticated Encryption with Length Expansion for Storage Devices*. If we refer to the description that the standard makes of himself, we have an idea of what we will find throughout this chapter:

The purpose of this standard is to describe a method of encryption for data stored in sector-based devices where the threat model includes possible access to stored data by the adversary. The standard specifies the encryption transform and a method for exporting/importing encryption keys for compatibility between different implementations. Encryption of data in transit is not covered by this standard.

This standard defines the XTS-AES tweakable block cipher and its use for encryption of sector-based storage. XTS-AES is a tweakable block cipher that acts on data units of 128 bits or more and uses the AES block cipher as a subroutine. The key material for XTS-AES consists of a data encryption key (used by the AES block cipher) as well as a "tweak key" that is used to incorporate the logical position of the data block into the encryption. XTS-AES is a concrete instantiation of the class of tweakable block ciphers described in [12]. The XTS-AES addresses threats such as copy-and-paste attack, while allowing parallelization and pipelining in cipher implementations.

After deeply analyze the process of Cryptographic Protection of Data on Block-Oriented Storage Devices, we will discuss some considerations of the different features included on the standard and could have some other approach.

3.1 Theory

This section is intended to unify all knowledge displayed throughout the previous chapter, to finally understand in-depth how the new encryption standard developed by IEEE works that is intended to provide security for tape storage by means of privacy and integrity.

IEEE 1619.1 specifies an architecture for protection of data in variable-length block storage media such as tape cartridge. It utilizes AES-GCM and AES-CCM with the 256-bit key size for privacy and integrity of data stored on tape.

3.1.1 Data units and tweaks

When we talk about data streams, we are talking about the data that we pretend to encrypt and safely store on the device, and this standard refers to the encryption of this data stream dividing it into consecutive equal-size data units. Divide the data unit is important because of the order of each data block: the order of each data block will be the tweak value that the standard will us to provide aleatory tho the cipher stream. Data that is not intended to be encrypted, it will not be considered data stream.

The minimum size of each data unit should be 128 bits and that will be the size of each block in which the data unit will be divided if it is greater than 128 bits. It could be possible that the last block of the data unit will be shorter than 128 bits, for this problem the standard implements the CipherText Stealing feature, we will see it deeply in following sections. There are some restrictions or recommendations with the number of 128-bit blocks: They should not be greater that $2^{128} - 2$ in the data unit and also they should not be greater than 2^{20}.

As we referred before, to each data unit a tweak value is assigned, based on its order, that is a non-negative integer. Those tweak values are assigned consecutively, starting from an arbitrary non-negative integer. That tweak value is converted into a little-endian byte array before could be encrypted using AES algorithm. For example, the tweak value $1a2b3c4d5e_{16}$ is converted to the byte array $[5e\ 4d\ 3c\ 2b\ 1a]_{16}$.

Is not an aim of the standard to describe how the mapping between the data unit and the transfer, placement and composition on the data stored device should be done. The different data storage devices that implement this standard, should include documentation specifying how this mapping is done within the device.

3.1.2 Multiplication by a primitive element α

During the encryption and decryption process, the result of the encryption of the 128-bit tweak with AES using Key_1, $E_{AES}(Key_1, i)$, is subsequently multiplied by the j-th power

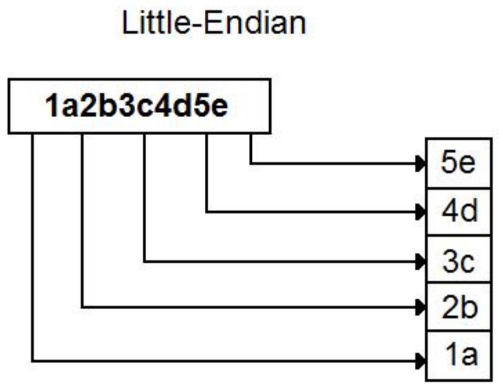


Figure 3.1: Little-Endian Transformation

of α, where α is a primitive element of $GF(2^{128})$.[1]

Before doing the multiplication process, the result of the encryption is firstly converted into a byte array $a_0[k]$, $k = 0, 1, ..., 15$, where $a_0[0]$ is the first byte of the AES block. The multiplication process is defined by the following elements:

Inputs	j is the power of α byte array $a_0[k]$, $k = 0, 1, ..., 15$
Outputs	byte array $a_j[k]$, $k = 0, 1, ..., 15$

Table 3.1: Inputs and outputs in the multiplication by a primitive element α procedure.

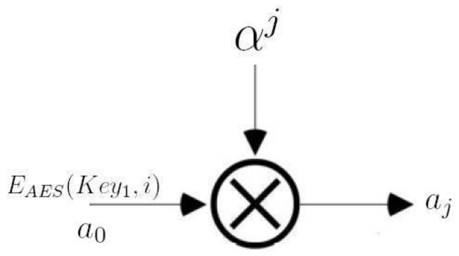


Figure 3.2: Multiplication by a primitive element

The output array $a_j[k]$ is calculated recursively using the following formulas where i is iterated from 0 to j:

$$a_{i+1}[0] \leftarrow (2(a_i[0] \; mod \; 128)) \oplus (135 \lfloor a_i[15]/128 \rfloor)$$

[1]If $\alpha \in GF(q)$ and the order of α is $n = q - 1$, then α is a primitive element of $GF(q)$.

$$a_{i+1}[k] \leftarrow (2(a_i[k]\, mod\, 128)) \oplus \lfloor a_i[k-1]/128 \rfloor, k = 1, 2, ..., 15$$

If we observe closely at the formulas, we could see that the bytes of the vector a_{i+1} are formed by combining the bytes of the vector a_i using the values of the current and previous byte, it could be considered a left shift. Also the final byte (15^{th}) of the vector a_i is used to form the first byte of the vector a_{i+1}, the value 135 that appears in the formula is derived from the modulus of the Galois Field (polynomial $x^{128} + x^7 + x^2 + x + 1$).[13]

3.1.3 XTS-AES encryption procedure

The aim of this section of the Thesis is to join all the ideas together and show how the encryption process is done. First we will analyze how the encryption of a 128-bit block is done, and after we will study the encryption process of a data stream with an arbitrary length.

Encryption of a single 128-bit block

The process of encryption XTS-AES system can be summarized by the following equation:

$$C \leftarrow E_{XTS-AES}(Key, j, i, P)$$

Where:

- Key is the 256 or 512 bit used during the XTS-AES encryption process. The Key is composed of the concatenation of two equal-sized keys that will be used for different purposes during the encryption process, $Key = Key_1 | Key_2$.
- j is the order of the 128-bit blocks inside the data unit.
- i is the 128-bit tweak value.
- P is the 128-bit block of plaintext.
- C is the 128-bit block resulting of the encryption process.

This equation refers to the final result of the encryption process, but for arriving to the final ciphertext it is necessary to follow a series of steps:

1. $T \leftarrow E_{AES}(Key_2, i) \otimes \alpha^j$
2. $PP \leftarrow P \oplus T$
3. $CC \leftarrow E_{AES}(Key_1, PP)$
4. $C \leftarrow CC \oplus T$”

As we can see in the figure 3.3, the encryption process starts by encrypting the tweak value i with the second part of the key Key_2. The result of this firstly encryption is then transformed using the $GF(2^{128})$ primitive element described in 3.1.2. α is this primitive element and it will vary with each data unit depending of j, the number of 128-bits in which it is divided the data unit. After that, this result, T, will be xor-ed with the 128-bit block of plaintext and then the second encryption take place, encrypting the result of $T \oplus P$ with the first part of the Key Key_1. Then T will be again xor-ed, this time with the result, CC, of the last AES encryption. The final result is the ciphertext C.

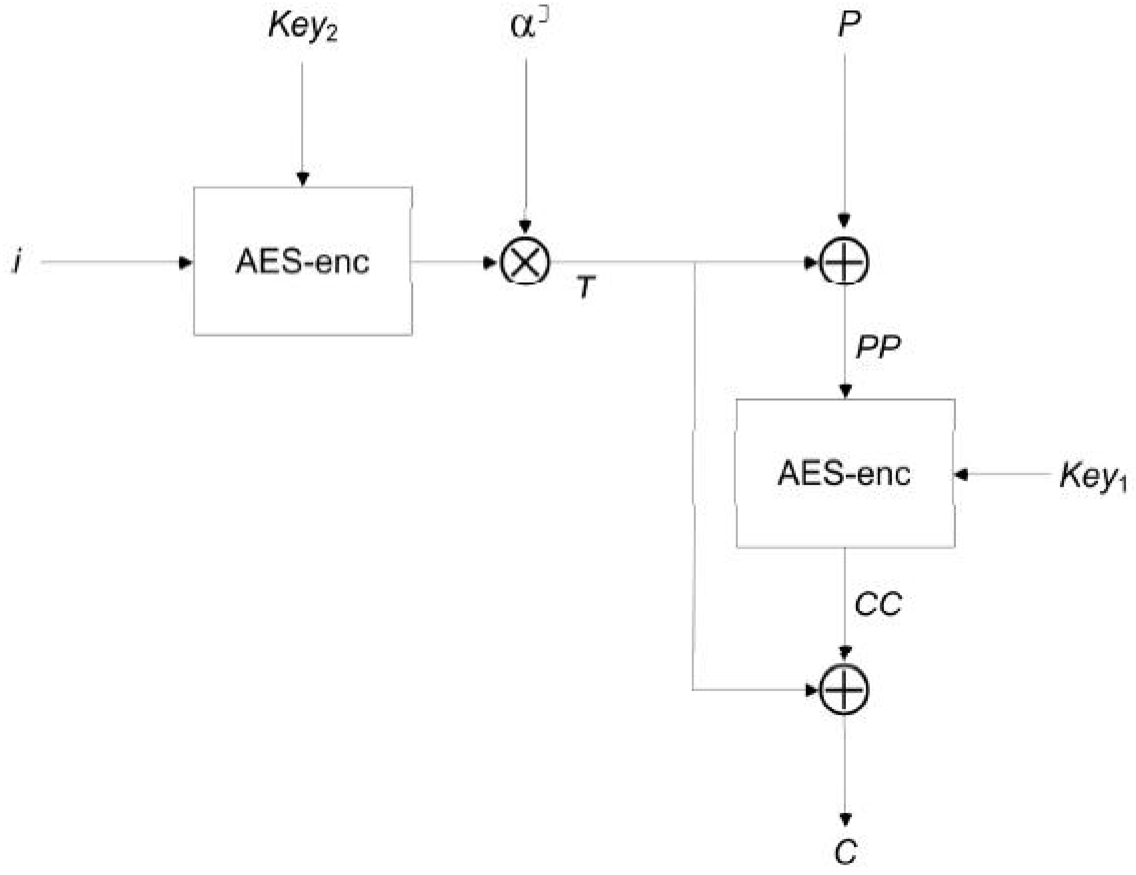


Figure 3.3: Diagram of the XTS-AES Encryption process [13]

XTS-AES encryption of a data unit

We have explained the encryption process of a 128-bit block of plaintext, the next step is to explain how to encrypt a data stream with an arbitrary stream. The equation that models this process is:

$$C \leftarrow E_{XTS-AES}(Key, i, P)$$

This equation is similar than the equation that model the encryption of a 128-bit block, the difference is the number of elements that compose the equations, the number of the 128-block is not included on the equation, but it will be used during the encryption process.

- Key is the 256 or 512 bit used during the XTS-AES encryption process. The Key is composed of the concatenation of two equal-sized keys that will be used for different purposes during the encryption process, $Key = Key_1|Key_2$.
- i is the 128-bit tweak value.
- P is the plaintext stream.
- C is the ciphertext. Same length as plaintext.

The data stream should be divided into 128-bit blocks. The length of the last block could not be 128 bit.

$$P = P_0|P_1|...|P_{m-1}|P_m$$

The length of $P_0, ..., P_{m-1}$ is 128 bit while the length of the last block P_m could be $0 < bit < 127$. The following steps explain how the plaintext is encrypted:

```
1. for $r \leftarrow 0$ to $m-2$ do
    (a) $C_r \leftarrow E_{XTS-AES}(Key, P_j, i, r)$
2. $b \leftarrow$ bit-size of $P_m$
3. if $b = 0$ then do
    (a) $C_{m-1} \leftarrow E_{XTS-AES}(Key, P_{m-1}, i, m-1)$
    (b) $C_m \leftarrow$ empty
4. else, if $b \neq 0$
    (a) $CC \leftarrow E_{XTS-AES}(Key, P_{m-1}, i, m-1)$
    (b) $C_m \leftarrow$ first $b$ bits of $CC$
    (c) $CP \leftarrow$ last $(128 - b)$ bits of $CC$
    (d) $PP \leftarrow P_m|CP$
```

   (e) $C_{m-1} \leftarrow E_{XTS-AES}(Key, PP, i, m)$

5. $C \leftarrow C_0|C_1|...|C_{m-1}|C_m$

Is interesting to note how the index of the 128-bit block is included in the encryption process. We saw in the equation that model the encryption of an arbitrary data stream that the order of the block is not taken into account. However, as we could see during the steps, the order of the 128-bit block is used to encrypt as is described before. The length of the last block is checked, if is 0 there is no problems: all blocks are 128-bits. If the last block is not empty then is necessary to use the CipherText Stealing (CTS) technique. As we could see in the step description some bits of the penultimate 128-bit block are used to form the two last block of the ciphertext. Figure 3.4 show a diagram of how the CTS technique in the encryption process is done.

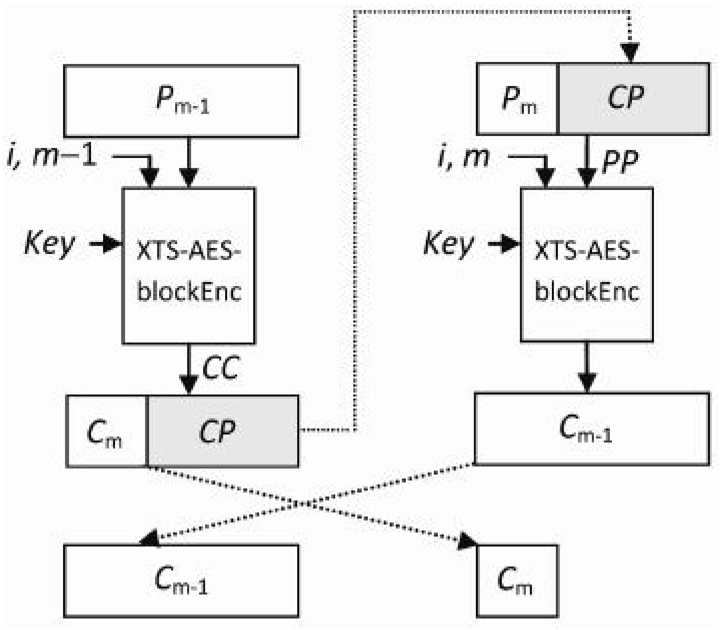


Figure 3.4: Encryption of last two blocks when last block is not empty [13]

As we see in the figure 3.4 the XTS-AES encryption of the penultimate 128-bit block of plaintext is done as the encryption of the rest of 128-bit blocks. Then the first b bits of the result are selected and form the last block of the ciphertext. The rest of the bits of CC are combined with the last block of plaintext (is not a 128-bit block) to form a 128-bit block. This new block is then encrypted using XTS-AES as the rest of the blocks. The result of this encryption will perform the penultimate block (the last 128-bit block) of the ciphertext.

3.1.4 XTS-AES decryption procedure

After explaining the encryption process of the standard, is time to see how the how the decryption process will be done. The encryption and decryption are similar but with some differences that we will se during this section. As the previous section, we will see the decryption of a single 128-bit block and then we will analyze the decryption of a cipher stream of an arbitrary length.

Decryption of a single 128-bit block

As the encryption process, we could also model the decryption process with an equation:

$$P \leftarrow D_{XTS-AES}(Key, i, j, C)$$

Where:

- Key is the 256 or 512 bit used during the XTS-AES decryption process. The Key is composed of the concatenation of two equal-sized keys that will be used for different purposes during the decryption process, $Key = Key_1|Key_2$.
- j is the order of the 128-bit blocks inside the data unit.
- i is the 128-bit tweak value.
- C is the 128-bit block of ciphertext.
- P is the 128-bit block resulting of the decryption process.

A series of steps is necessary to follow to finally decrypt the 128-bit block of ciphertext:

1. $T \leftarrow E_{AES}(Key_2, i) \otimes \alpha^j$
2. $CC \leftarrow C \oplus T$
3. $PP \leftarrow D_{AES}(Key_1, CC)$
4. $P \leftarrow PP \oplus T$"

The following figure shows the process of decryption of a single 128-bit block.

As we can see in the figure 3.5, the decryption process starts by encrypting the tweak value i with the second part of the key Key_2. The result of this firstly encryption is then transformed using the $GF(2^{128})$ primitive element described in 3.1.2. α is this primitive element and it will vary with each data unit depending of j, the number of 128-bits in which it is divided the data unit. After that, this result, T, will be xor-ed with the 128-bit block of ciphertext and then the decryption procedure take place, decrypting the result of $T \oplus C$ with the first part of the Key Key_1. Then T will be again xor-ed, this time with the result, PP, of the AES decryption. The final result is the recovered plaintext P.

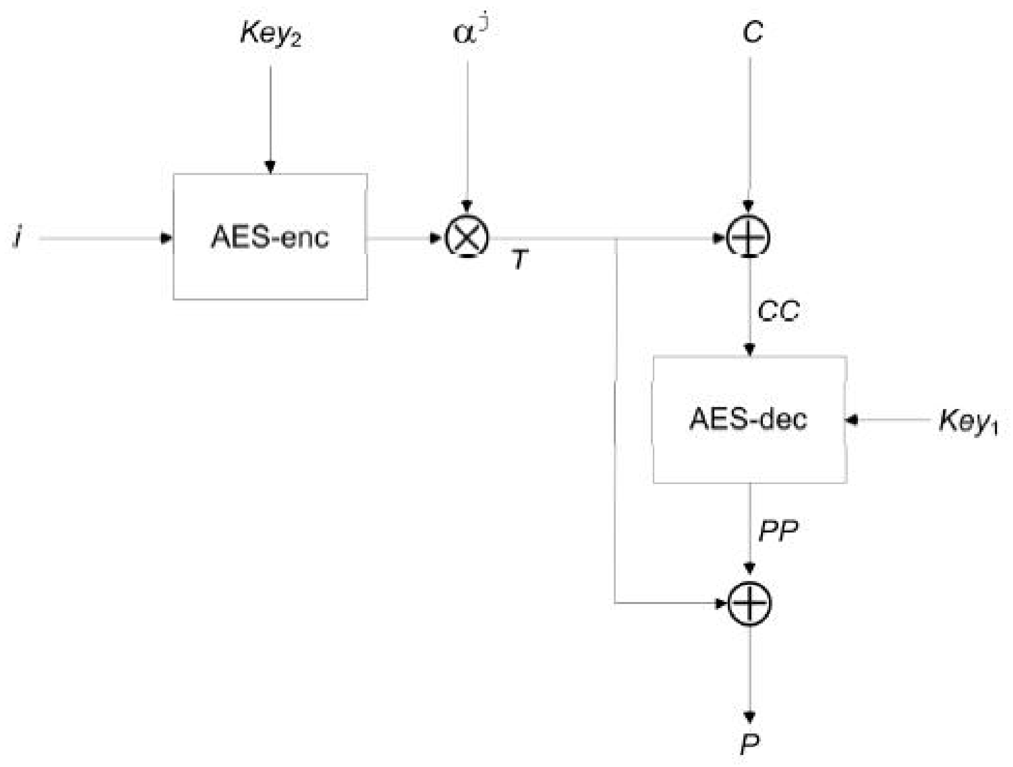


Figure 3.5: Diagram of the XTS-AES Decryption process [13]

In this figure we observe the similarity between the encryption and decryption processes. If we compare both 3.3 and 3.5 figures, we observe that the unique differences between them are position exchange between the plaintext and the ciphertext, and the second XTS-AES block that in this case is used as decryption mode. However the first XTS-AES block is used as encryption mode, that means that the tweak value is also encrypted as in the encryption process.

XTS-AES decryption of a data unit

As we did with the encryption process, the next step to deeply understand the decryption mode is to analyze how the decryption of a ciphertext stream with an arbitrary length is done.The equation that models this process is:

$$P \leftarrow D_{XTS-AES}(Key, i, C)$$

Again, the order of each 128-bit block is not used to model the complete decryption because each 128-bit block will has its own order j and does not depend on what data unit are we trying to encrypt but the data unit size.

- Key is the 256 or 512 bit used during the XTS-AES decryption process. The Key is composed of the concatenation of two equal-sized keys that will be used for different purposes during the decryption process, $Key = Key_1|Key_2$.
- i is the 128-bit tweak value.
- C is the ciphertext stream.
- P is the plaintext resulting of the decryption process, same size as ciphertext.

The data stream should be divided into 128-bit blocks. The length of the last block could not be 128 bit.

$$C = C_0|C_1|...|C_{m-1}|C_m$$

The length of $C_0, ..., C_{m-1}$ is 128 bit while the length of the last block C_m could be $0 < bit < 127$. The following steps explain how the ciphertext is decrypted:

```
1. for $r \leftarrow 0$ to $m-2$ do
    (a) $P_r \leftarrow D_{XTS-AES}(Key, C_j, i, r)$
2. $b \leftarrow$ bit-size of $C_m$
3. if $b = 0$ then do
    (a) $P_{m-1} \leftarrow D_{XTS-AES}(Key, P_{m-1}, i, m-1)$
    (b) $P_m \leftarrow$ empty
4. else, if $b \neq 0$
    (a) $PP \leftarrow D_{XTS-AES}(Key, C_{m-1}, i, m-1)$
    (b) $P_m \leftarrow$ first $b$ bits of $PP$
    (c) $CP \leftarrow$ last $(128 - b)$ bits of $PP$
    (d) $CC \leftarrow C_m|CP$
```

(e) $P_{m-1} \leftarrow D_{XTS-AES}(Key, CC, i, m)$

5. $P \leftarrow P_0|P_1|...|P_{m-1}|P_m$

The procedure to recover the last two block of ciphertext, is similar than the procedure executed in then encryption process. Is interesting to note how the index of the 128-bit block is included in the decryption process. We saw in the equation that model the decryption of an arbitrary data stream that the order of the block is not taken into account. However, as we could see during the steps, the order of the 128-bit block is used to decrypt as is described before. The length of the last block is checked, if is 0 there is no problems: all blocks are 128-bits. If the last block is not empty then is necessary to use the CipherText Stealing (CTS) technique. As we could see in the step description some bits of the penultimate 128-bit block are used to form the two last block of the plaintext. Figure 3.6 show a diagram of how the CTS technique in the decryption process is done.

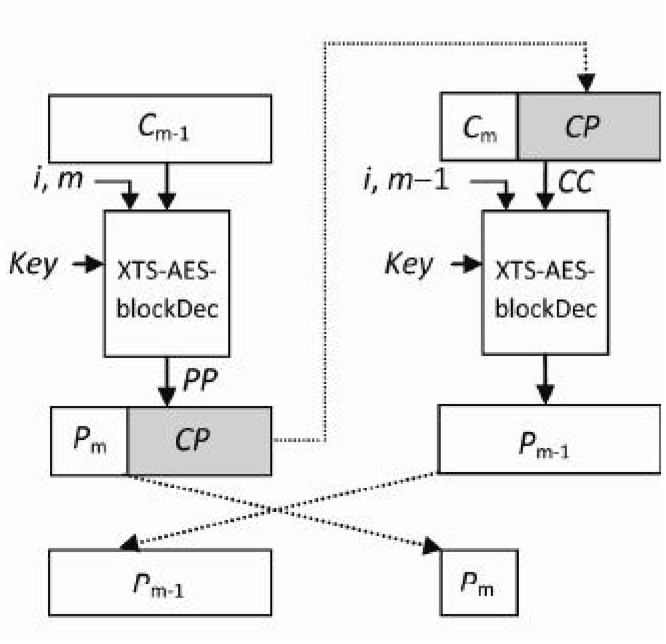


Figure 3.6: Decryption of last two blocks when last block is not empty [13]

As we see in the figure 3.6 the XTS-AES decryption of the penultimate 128-bit block of ciphertext is done as the decryption of the rest of 128-bit blocks. Then the first b bits of the result are selected and form the last block of the ciphertext. The rest of the bits of PP are combined with the last block of ciphertext (is not a 128-bit block) to form a 128-bit block. This new block is then decrypted using XTS-AES as the rest of the blocks. The result of this decryption will perform the penultimate block (the last 128-bit block) of the plaintext.

3.2 Comments on XTS-AES

After P1619 Task Group of the Security in Storage Working Group (SISWG) of the Institute of Electrical and Electronics Engineers, Inc. (IEEE) had submitted the XTS-AES algorithm to NIST as an encryption mode of operation of the Advanced Encryption Standard (AES) block cipher, NIST opened a period (from June 5, 2008 to September 3, 2008) to express public comments related to the XTS-AES algorithm. During this period the extract of IEEE standard 1619-2007 was available for free, after that period the standard was available paying a charge. That decision was unpopular and one of the topics that NIST suggested to discuss. Those are the topics that NIST proposed to discuss in the comments:
.

- The XTS-AES algorithm itself.
- The depth of support in the storage industry for which it was designed.
- The appeal of XTS for wider applications.
- The proposal for the approved specification to be available only by purchase from IEEE.
- Concerns of intellectual property rights.

Within this period, NIST received different emails and public documents analyzing the different topics proposed above from particular people interested on the standard like P. Rogaway [14] or Moses Liskov and Kazuhiko Minematsu [15], and also from people that belong enterprises as Microsoft Corporation or Entropic communications.

Along the following sections, the different topics proposed by NIST are going to be developed and analyzed, explaining what are the different opinions submitted to NIST.

3.2.1 The XTS-AES algorithm itself

For Moses Liskov and Kazuhiko Minematsu (see [15]) there are two fundamental aspects in the standard that should be improved or clarified: First is the use of two keys instead of one, because the XTS mode of operation is a derived mode of XEX introduced by Rogaway which is proven the efficiency and safety of using a single key. The second is about the security of XTS-AES algorithm, is not sufficiently developed or proved.

Vijay Bharadwaj and Neils Ferguson of Microsoft Corporation expose some relevant aspect that should be treated to clearly define the security and functions of this new algorithm:

- Unclear Security Goals: They explain that the proposal does not clearly define what are the application-level security goals that XTS-AES wants to achieve, this made difficult to analyze and determine how widely applicable XTS-AES should be. SISWG explain that instead the proposal provide a good argument on the security of XTS-AES, they should have added more details about the applications that are available for XTS-AES.

- Temporal effects: It is possible for an attacker observe a disk for enough period of time and then take a significant advantage about the encryption mode used. SISWG recognize that it could be a kind of attack that could suffer not only XTS-AES mode but some other modes of operation, but this seems impracticable.

- Attack on large data units: Large data units weak the system, for that reason the standard should not allow data units larger than 2^{20} as is specified on the standard.

- Ciphertext Manipulation Attacks: XTS-AES is prone to fine-grained ciphertext manipulation attacks due works with 128-bit blocks. SISWG defends XTS-AES, instead admitting that an attacker has greater malleability with 128-bit narrow block encryption mode like XTS-AES than with wide-block encryption mode like XCB, arguing that XTS-AES is appropriate to systems where the attacker has limited access in making malicious attacks and the programs could detect a randomized 128-bit block with high probability.

For Phillip Rogaway, there are two idea that need clarification:

- A description of what security property of XTS-AES is supposed to deliver, specifically the CTS method. Although XTS is a version of XEX mode described by Rogaway, he does not oppose to it if the security is well defined. SISWG answer suggesting that XTS-AES is enough secure for applications that use it and also NIST wants to incorporate a similar CTS mode to the established CBC.

- XTS is not as strong as PRP, it could be a good choice only if is too hard to afford the computation or latency associated to computing a strong PRP. SISWG argue that there are many applications that could not support the extra computing effort that PRP implies, and there are also secure with XTS because of the fact that the attacker has limited access to the ciphertext.

Another person that submitted comments to NIST in order to debug the proposal of this standard is Boaz Shahar from Entropic Communications. In his opinion, a multiplication over $GF(2^{128})$ in Little Endian notation not only increases the complexity on the implementation and also it change NIST policy of using Big Endian notation. For SISWG there is not a problem using Little Endian in order is optimus for software based on Little Endian systems.[2]

[2]Nowadays Little Endian systems is the most popular byte-ordering scheme.

3.2.2 The depth of support in the storage industry for which it was designed

Although the mode XTS-AES encryption is subject to a broad debate to discuss issues about their safety and possible improvements, various products and companies have shown interest and have implemented the XTS-AES mode among its features. As of June 2009 the following list of companies and products presented (or intended) XTS-AES as one of its features: AMCC, Bloombase Technologies, Brocade, CipherMax, DiskCryptor, EMC, Emulex, FreeOTFE, Freescale Semiconductor, GED-i, Helion, Hifn, Oxford Semiconductors, PMC-Sierra, SafeNet, Thales, TrueCrypt, WinMagic.[3]

Hard Disk vendors and IP Cores manufactures also include XTS-AES as one of its working modes as for example:

- Quantum, HP, IBM, Tandberg's LTO-4 Tape Drive uses IEEE 1619.1 GCM-AES.
- IBM's TS1120 Enterprise Tape Drive uses GCM.
- Sun's T10000 Enterprise Tape Drive uses 1619.1 CCM.

3.2.3 The appeal of XTS for wider applications

For David Clunie (See [14]) there ara applications as medical image exchange where XTS-AES mode can have a wide field of work due to the lack of standardization that exists and the need to store and transmit securely.

A very different point of view is the expressed in [17] by Michael Willett from Seagate Technology. Under his trial, XTS-AES presents different aspects that the implementation for more issues will not constitute a good idea:

- Interoperability: Many of the new storage units have their own internal encryption, so it would not be possible to combine the platters to use XTS-AES encryption mode.
- Archived data recovery in the distant future: It has not a hopeful future.
- There is a wide list of issues that may be considered before propose the establishment of XTS-AES encryption mode to other applications: Implementation errors should be avoided, XTS applicability, protection of debug ports, etc.

SISWG compare XTS operation mode with CBC, arguing that the complexity is roughly in both operation modes but the improvement of XTS is the less malleability of the ciphertext. It is true that there are more aspects that make secure an encryption mode and they are needed to accomplish before exporting XTS-AES to other application, but they also exist for other encryption modes.

[3]For a deeply description of how each companies and products implements XTS-AES, see [16].

3.2.4 The proposal for the approved specification to be available only by purchase from IEEE

This issue has united under a single opinion to all who have sent their comments to NIST. Because the measure adopted by the IEEE to allow the access to the standard under paying a fee has not been well received. The general opinion rejects as unacceptable pay to acquire the standard, and even more when they have been exposed it to debate to gather different points of view outside the IEEE. Many voices call on NIST that the access to the standards should be without any cost and should seek other forms of financing.

For its part SISWG defends itself arguing that, instead is a logical position, the development of the standard implies some cost and so it is necessary to pay a certain amount of money to acquire it, even it is not one of their competences but the IEEE. SISWG refers to NIST the decision to grant the access to standards, but while this is not possible should be consumers who paid the costs.

3.2.5 Concerns of intellectual property rights

The vast majority of people who have analyzed and commented on the operation mode XTS-AES agree that it is too early for standardization, and that the IP can be compromised if NIST accept to adopt the proposal as a standard because it would imply a mandatory implementation in all storage devices without the appropriate guarantees of security.

In response, SISWG suggests that there is ambiguity as to the rights of IP concerns, and that ambiguity affects all systems and modes of encryption, so this should not be a problem for the standardization.

Chapter 4

Tools: Software and Hardware

The aim of this chapter is to introduce briefly the tools used to probe the new standard published by IEEE that is the objective of this Thesis. We will see the different software used explaining the characteristic and how they implements the XTS-AES encryption mode, this software is:

- TrueCrypt
- FreeOFTE
- DiskCryptor

In the following chapter we will present the results obtained comparing the performance of the different software tools in two different machines with different characteristics: Operation System, memory, processor... This two computers are:

- Laptop Toshibas Satellite A220
- Desktop DELL

At the end of the chapter, a section about commercial hardware, the IP cores, is included. We will explain what are they, hoe implement the XTS-AES encryption mode and how could be useful.

4.1 Software

In this section, we will briefly introduce to the reader the different tools we have used to test XTS-AES encryption mode developed along the standard under study. It will be a short description of the characteristics of each software, in the next chapter we will show the different methods to provide security to the stored data using XTS-AES encryption.

4.1.1 TrueCrypt

TrueCrypt is the first software tool we are going to present. It is a great tool for encrypting data stored to avoid that an outsider can access them or disclose their content. The official website is http://www.truecrypt.org/ and presents the option to download different versions of the encryption tool depending the Operation System. We could find in the website a great support, both in tutorials and in various forums. The download also includes a complete PDF manual in English.

TrueCrypt is an excellent free source code software to encrypt and hide data on your PC that you believe it reserved using different encryption algorithms like AES, Blowfish, CAST5, Serpent, Triple DES, and Twofish, or a combination of some of them. TrueCrypt also supports AES with XTS mode with 256-bit key.

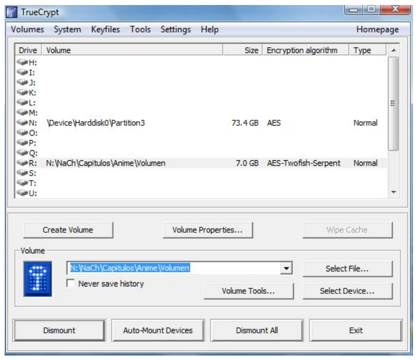
What it does is create a TrueCrypt ”secret volume”, which consists of a file that can have any name and that TrueCrypt can mount as a drive, with its respective identification according to the operating system installed. The contents of this file has its own filesystem and everything needed to operate as a common storage unit. What is recorded in the virtual drive is encrypted using technology and with powerful encryption mode you choose. When ”mounted” drive through TrueCrypt, it calls for the password that the user chose when creating this secret file. Encodes and decodes data on your hard disk in real time (”on the fly”).

Ideal for data you have saved on the encrypted drive can not be read without the password or encryption key, until that moment happens, the encrypted data will appear as a series of meaningless characters.

Apart from allowing to create an encrypted file, it allows you to encrypt an entire disk or partition and also the partition where is installed the Windows operating system, hidden volumes, etc. These options are only recommended for users with advanced knowledge and under its responsibility. The encrypted file can then be transported for example on a DVD, CD, USB memory, etc...

The following figures 4.1 and 4.2 show the main window of TrueCrypt with two encrypted file mounted and decrypted and how the system recognize them and we could have access to both decrypted files and read the information stored inside. In the figures we could see two different volumes decrypted. The first volume is a partition encrypted with AES, and the other volume is a container file stored inside the first volume and encrypted using three different techniques in cascade.

The following figures 4.1 and 4.2 show the main window of TrueCrypt with two encrypted file mounted and decrypted and how the system recognize them and we could have access to both decrypted files and read the information stored inside. In the figures we could see two different volumes decrypted. The first volume is a partition encrypted with AES, and the other volume is a container file stored inside the first volume and encrypted using three different techniques in cascade.



Drive	Volume	Size	Encryption algorithm	Type
H:				
I:				
J:				
K:				
L:				
M:				
N:	\Device\Harddisk0\Partition3	73.4 GB	AES	Normal
O:				
P:				
Q:				
R:	N:\NaCh\Capitulos\Anime\Volumen	7.0 GB	AES-Twofish-Serpent	Normal
S:				
T:				
U:				

Volume: N:\NaCh\Capitulos\Anime\Volumen

Figure 4.1: TrueCrypt main window

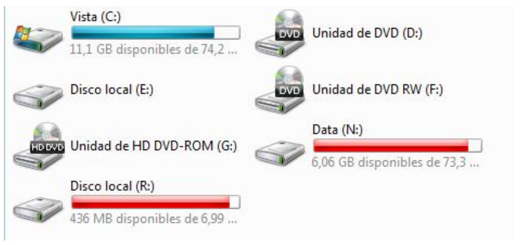

Vista (C:)
11,1 GB disponibles de 74,2 ...
Unidad de DVD (D:)
Disco local (E:)
Unidad de DVD RW (F:)
Unidad de HD DVD-ROM (G:)
Data (N:)
6,06 GB disponibles de 73,3 ...
Disco local (R:)
436 MB disponibles de 6,99 ...

Figure 4.2: Mounted volumes after TrueCrypt decryption

We will explain in 5.1 how to make the encryption of a partition and how to create a container with XTS-AES.

TrueCrypt includes a benchmark tool, with which is possible to test the performance of XTS-AES algorithm on both machines depending on the buffer size. The benchmark tool were executed in both machines with the following results:

Buffer Size (MB)	Encryption Speed (MB/s)	Decryption Speed (MB/s)	Mean Speed (MB/s)
0.1	115,67	98	107
0.5	117,67	110	114
1	111,33	114	112,67
5	103,1	104,63	103,7
10	108,33	112,67	110,67
50	100,93	106	103,33
100	107	107,33	107
200	105,67	104,03	104,97
500	109,33	108,67	108,67
1000	106,33	106	106,33

Table 4.1: Machine 1 XTS-AES benchmark

Speed is more constantly in Machine 2 (instead of a little variation when the buffer size is 5 MB due to CPU load), but Machine 1 achieves higher speed.

Buffer Size (MB)	Encryption Speed (MB/s)	Decryption Speed (MB/s)	Mean Speed (MB/s)
0.1	79.8	78.4	79.1
0.5	80.1	78.1	79.1
1	80.3	77	78.6
5	75	66.2	70.6
10	79.5	77.8	78.7
50	78.7	77	77.8
100	79.8	78.1	79
200	79.5	78.1	78.9
500	79.4	78.2	78.8
1000	79	77.6	78.3

Table 4.2: Machine 2 XTS-AES benchmark

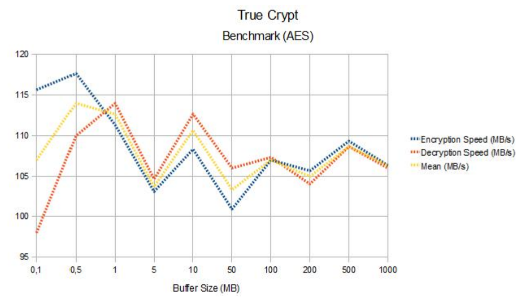


Figure 4.3: Machine 1 XTS-AES benchmark

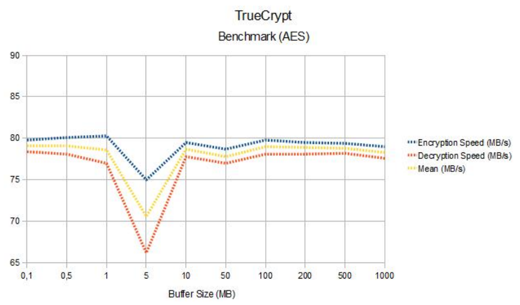


Figure 4.4: Machine 2 XTS-AES benchmark

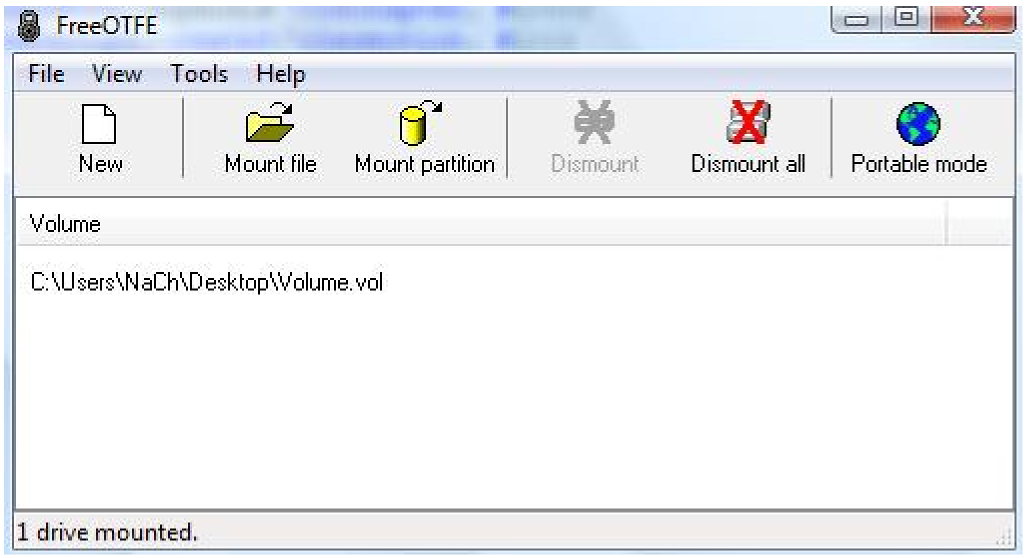
4.1.2 FreeOTFE

The following encryption tool we are going to present and it also implements the XTS-AES encryption mode is FreeOTFE[1]. In the website http://www.freeotfe.org/ is possible to download the installer and read the user manual and the FAQ to learn how to use it.

Under the trade name FreeOTFE we find a file encoder ”on-the-fly” for PCs with Windows2000/XP/Vista and devices with Windows Mobile 2003/2005 and Windows Mobile 6. FreeOTFE is an open source program that creates a new virtual disk drive in which we can store encrypted confidential file, simply storing it in the unit.

The program has a ”wizard” who is responsible for guiding us in building our encrypted unity. Supports multiple encryption algorithms: AES, Twofish, Blowfish or Serpent can use functions hash like MD5, SHA-512 or Tiger, among others. The virtual drive can be created by a file, for example by allocating a folder on the memory card, or use a partition. FreeOTFE has many functions and options for achieving a virtual safe for our data.

FreeOTFE works perfectly on Windows XP with NTFS file system. For filesystems FAT/FAT32 maximum volume that can be created is 4 GB, in the NTFS system there is almost no limit: supports volumes of more than 10 million GB. In Windows Vista exists a problem caused by the unpopular UAC, if is turned on will not allow the normal performance of FreeOTFE, one solution is saving the application elsewhere in the system outside ”Program Files”.



C:\Users\NaCh\Desktop\Volume.vol

1 drive mounted.

Figure 4.5: FreeOTFE main window

This figure shows the main window of FreeOTFE encryption tool. We can observe one volume mounted, this volumen is encrypted using XTS-AES with a 256-bit key size. We will se how to create volumes in section 5.1.

[1]FreeOTFE is the short form of Free On The Fly Encryption

FreeOTFE presents also a portable version, allowing encrypt an entire disk or flash memory. It also gives us the ability to create a volume on an invisible disc, i.e., for example the C disk could contain a new 20 GB drive will only be visible by running the application and of course, entering the key or password. As at the same time we have the portable version, we can copy it in a flash memory and encrypt the entire contents or create a volume invisible inside the flash memory.

4.1.3 DiskCryptor

The last encryption tool that will help us to test the XTS-AES encryption mode is DiskCryptor. In the website http://www.diskcryptor.de/en/ we can download the installer file and read the manuals to learn how to use it. DiskCryptor is different than TrueCrypt and FreeOTFE because DiskCryptor only allows to encrypt a disk partition or the entire disk.

DiskCryptor is a free application that will encrypt the hard disk partitions that you choose. Its size is quite small and installation is fast. The encryption system used is AES, Twofish, Serpent, Twofish-AES, Twofish-Serpent, Serpent-AES or AES-Twofish-Serpent in XTS mode. XTS Mode is specially designed to encrypt and protect disks against any attack.

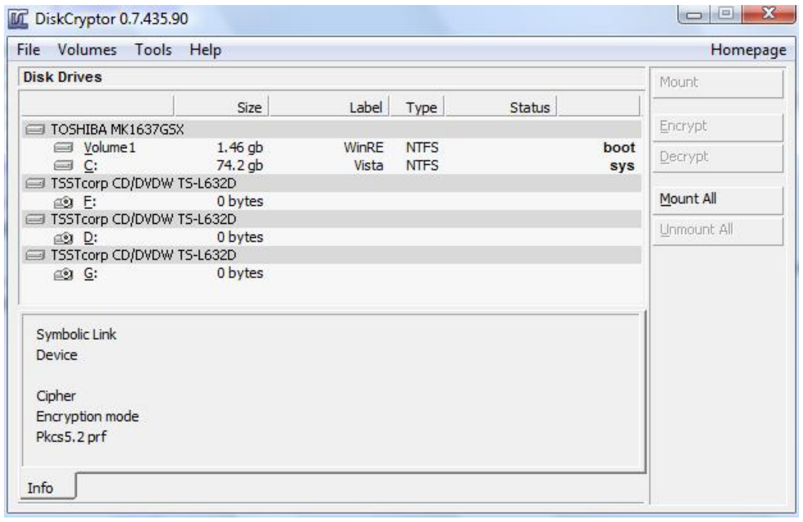
The encryption key is randomly created and saved in the first sector of the hard disk. DiskCryptor is under the GPLv3 license, works for:

- Windows 2000 SP0-SP4
- Windows XP (x86, x64) SP0-SP3
- Windows Server 2003 (x86, x64) SP0-SP2
- Windows Vista SP0, SP1
- Windows Vista x64 SP0, SP1
- Windows Server 2008
- Windows Server 2008 x64

DiskCryptor is a free project developed to provide reliable protection on the computer, using a strong encryption system that prevents access and modification on any disk partition.

The program is able to apply this condition on the file systems FAT32 and NTFS, even if they correspond to the unit that is installed in the operating system. The encryption process is very simple, just choose the partition to protect and define the password that will allow the release of same.

DiskCryptor will display a list all available drives on the computer, adding additional information concerning the name of the volume, storage capacity, file system and state. If we are looking for an effective way to prevent the unauthorized use of your content, this application is the perfect solution.



	Size	Label	Type	Status	
TOSHIBA MK1637GSX					
Volume1	1.46 gb	WinRE	NTFS		boot
C:	74.2 gb	Vista	NTFS		sys
TSSTcorp CD/DVDW TS-L632D					
F:	0 bytes				
TSSTcorp CD/DVDW TS-L632D					
D:	0 bytes				
TSSTcorp CD/DVDW TS-L632D					
G:	0 bytes				

Symbolic Link
Device

Cipher
Encryption mode
Pkcs5.2 prf

Figure 4.6: DiskCryptor main window

4.2 Hardware

This section will show briefly the two different machines we have used to test the encryption software. These machine are completely different between them and we will realize different test on them.

4.2.1 Laptop Toshiba Satellite A200

The first machine is a Toshiba Laptop. Is more powerful than the other computer and presents the following features:

Company	TOSHIBA
Model	Satellite A200
Processor	Intel Core 2 Duo, T7100 @ 1.80 GHZ
Operating System	Windows Vista Home Premium with Service Pack 2
RAM	2 GB
Hard Disk	160 GB
Partitions	2

Table 4.3: Toshiba laptop features

4.2.2 Desktop DELL

THe other machine in which we have tested the different encryption encryption software is a DELL desktop. It presents the following features:

Company	DELL
Model	OPTIPLEX GX280
Processor	Pentium 4, 3 GHZ
Operating System	Windows XP Professional Version 2002 with Service Pack 3
RAM	1.49 GB
Hard Disk	75 GB
Partitions	1

Table 4.4: Dell desktop features

4.3 Commercial Hardware: IP cores

General description

What is an IP core?

Within a FPGA may include the functionality of several integrated circuits. This functionality can be developed by the same team or acquired through a third party. Because these features are like electronic components, but without their physical part, they are often called virtual components. In industry they are known as block intellectual property or IP cores.

The families XTS2 and XTS3 implemments the XTS-AES encryption mode defined by IEEE.

XTS3 implements the NIST standard AES cipher in the XEX/XTS mode for encryption and decryption. The XTS3 family of cores covers a wide range of area/throughput combinations, allowing the designer to choose the smallest core that satisfies the desired clock/throughput requirements. XTS2 is similar to XTS3, but supports only 128-bit keys. Each core contains the base AES core AES1 and is available for immediate licensing.[18] [19]

The XTS3 family fully supports the XTS-AES algorithm with key size of 256 and 128 bits. XTS2 family supports only XTS-AES with key size of 128.

Key features

The XTS2 and XTS3 families implements some features to manage the Key:

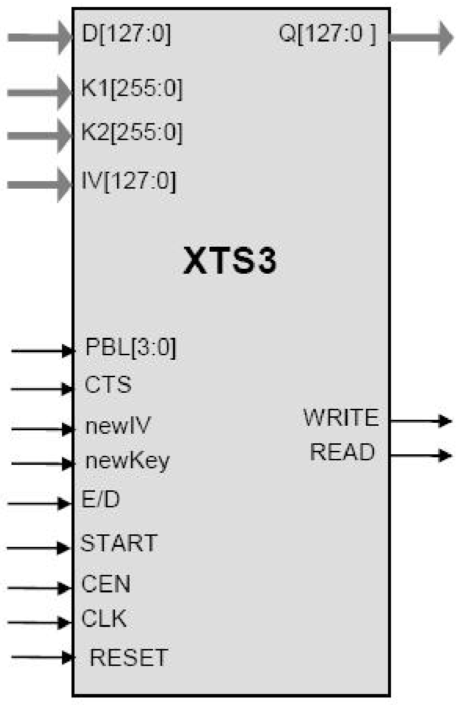


Figure 4.7: XTS3 IP core symbol.[18]

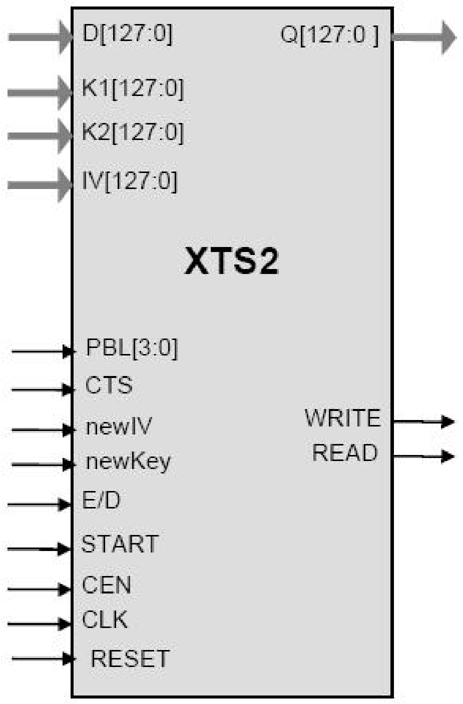


Figure 4.8: XTS2 IP core symbol.[19]

- Key Features Small size: XTS2-12.8 starts at less than 30,000 ASIC gates and delivers throughput of 7 Gbps. XTS3-18.2 starts at less than 50,000 ASIC gates at throughput of 18.2 bits per clock. The fastest cores in the families, XTS2-128 and XTS3-128, deliver 128 bits of throughput per clock (for example, at 500 MHz clock the maximum throughput is 64 Gbps).
- Completely self-contained: does not require external memory.
- Supports both encryption and decryption.
- Includes key expansion and CTS support.
- Supports XEX-based Tweaked CodeBook mode (TCB) with CipherText Stealing (CTS) (XTS) mode encryption and decryption.
- $128 + 128$ and $256 + 256$ bit AES keys supported.
- Easily parallelizable for even higher data rates.

There are more companies that develop those type of FPGA: IP cores with XTS-AES encryption mode support. We have only described IP cores from ”IP Cores, Inc.” as en example of commercial hardware that implements the new encryption mode described in IEEE 1619.1 and it is the aim of this Thesis.

Name	Type	Description
CLK	Input	Core clock signal
RESET	Input	HIGH level asynchronously resets the core
CEN	Input	Synchronous enable signal. When LOW the core ignores all its inputs and all its outputs must be ignored.
E/D	Input	When HIGH, core is encrypting, when LOW core is decrypting When HIGH, core uses the 256-bit key
K256	Input	When HIGH, core uses the 256-bit key
START	Input	HIGH level starts the input data processing
READ	Output	Read request for the input data byte
WRITE	Output	Write signal for the output interface
D[127:0]	Input	Input Data (other data bus widths are also available) plain or cipher text
IV[127:0]	Input	IV (logical position)
K1[255:0]	Input	AES key
K2[255:0]	Input	Tweak key (Key_2)
Q[127:0]	Output	Output plain or cipher text
newKey	Input	New AES Key available on K1 input
newIV	Input	New Tweak Key available on K2 input, and new IV available on IV input
CTS	Input	Marks the last full 128-bit block of the data unit in case that the next block of this data unit is less than 128 bit (CTS mode)
PBL[3:0]	Input	Partial Block length (in bytes)

Table 4.5: IP core PIN description.[18] [19]

Chapter 5

Methods and results

In the previous chapter, we briefly introduced each of the encryption tools that will help us to analyze the functionality of the new standard approved by the IEEE and its the main object of this Thesis. Throughout this chapter, we will take a look for different methods of protecting data stored on a device that offer each of the tools used such create encrypted containers to secure store files, encrypt partitions or encrypt the entire system.

After knowing the different ways to use the XTS-AES encryption presented in the standard, we will try to compare the way of carrying it out by each of the tools used and then we will analyze the time response of each of them, its functionality, safety, simplicity, etc...

5.1 Methods

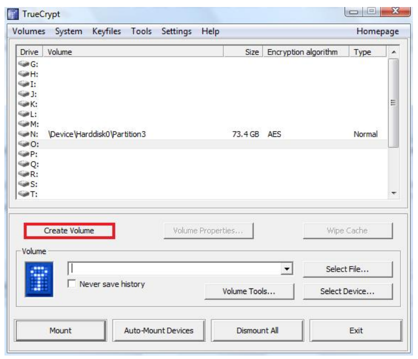
5.1.1 Container

What do we mean by the word ”container”? Programs such TrueCrypt or FreeOTFE offer us the possibility to create volumes of certain size, thus protected by the XTS-AES encryption and authentication, in order to securely store files in them, we will refer them as ”containers”.

We will explain the necessary steps that we must follow for the creation of such containers and store files in them and how these files are secure and inaccessible from the outside by someone ignorant of the key. We will also try to decrypt these containers with other tools to achieve read the files inside.

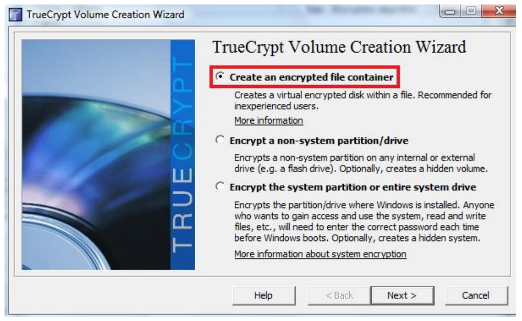
TrueCrypt

For creating a new container, we must firstly click the button ”Create Volume” (fig 5.1) to launch the volume creation process. Once the Wizard is launched, we will find three options in the first step, we are going to select the first of them because we want to see how create new containers (fig 5.2). We have the possibility to create them hidden, for simplicity we will explain how to create a standard container.



Drive	Volume	Size	Encryption algorithm	Type
G:				
H:				
I:				
J:				
K:				
L:				
M:				
N:	\Device\Harddisk0\Partition3	73.4 GB	AES	Normal
O:				
P:				
Q:				
R:				
S:				
T:				

Figure 5.1: TrueCrypt Step 1



TrueCrypt Volume Creation Wizard

[✓ Create an encrypted file container]

Creates a virtual encrypted disk within a file. Recommended for inexperienced users.

More information

Encrypt a non-system partition/drive

Encrypts a non-system partition on any internal or external drive (e.g. a flash drive). Optionally, creates a hidden volume.

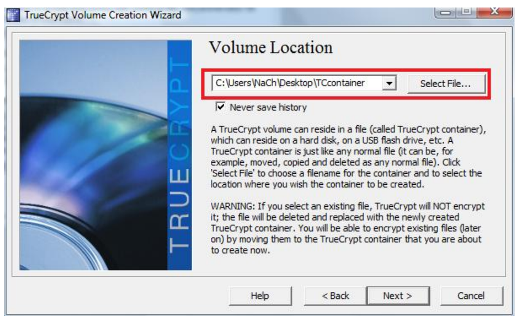
Encrypt the system partition or entire system drive

Encrypts the partition/drive where Windows is installed. Anyone who wants to gain access and use the system, read and write files, etc., will need to enter the correct password each time before Windows boots. Optionally, creates a hidden system.

More information about system encryption

Figure 5.2: TrueCrypt Step 2

Afterwards, is necessary specify the path and the name where the container will be created (fig 5.3) and the algorithm and the hash that we are going to use to protect the stored data. We will choose AES encryption algorithm (internally it will work in XTS mode) and SHA-512 as hash algorithm (fig 5.4).



Volume Location

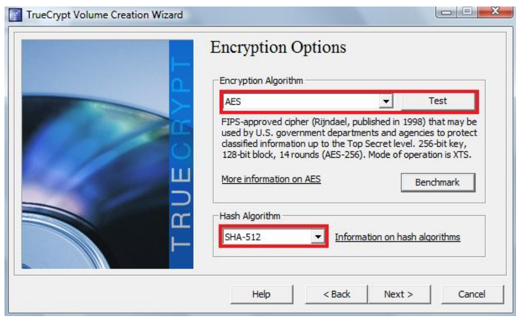
C:\Users\NaCh\Desktop\TCcontainer

✓ Never save history

A TrueCrypt volume can reside in a file (called TrueCrypt container), which can reside on a hard disk, on a USB flash drive, etc. A TrueCrypt container is just like any normal file (it can be, for example, moved, copied and deleted as any normal file). Click 'Select File' to choose a filename for the container and to select the location where you wish the container to be created.

WARNING: If you select an existing file, TrueCrypt will NOT encrypt it; the file will be deleted and replaced with the newly created TrueCrypt container. You will be able to encrypt existing files (later on) by moving them to the TrueCrypt container that you are about to create now.

Figure 5.3: TrueCrypt Step 3


TrueCrypt Volume Creation Wizard
TRUECRYPT
Test
Benchmark
Information on hash algorithms
Help
< Back
Next >
Cancel

Encryption Options

Encryption Algorithm: AES

FIPS-approved cipher (Rijndael, published in 1998) that may be used by U.S. government departments and agencies to protect classified information up to the Top Secret level. 256-bit key, 128-bit block, 14 rounds (AES-256). Mode of operation is XTS.

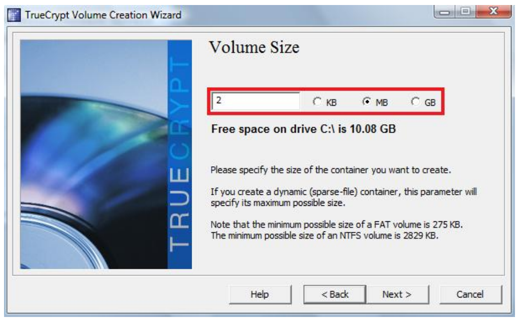
More information on AES

Hash Algorithm: SHA-512

Figure 5.4: TrueCrypt Step 4

After selecting path, name and encryption algorithm for the container, we must to specify the size we want to have the container (fig 5.5) and then introduce the password (fig 5.6). Is it possible to include a file as a password selecting the option ”use key files”, but we will introduce manually the password. This password will serve as an authentication method to recover the stored data on the container.

We can now finally create the container clicking ”Format”. Before that, in order to introduce randomization to the encrypted data, we should move the mouse within the Wizard



Volume Size

2 ( ) KB [✓ MB] ( ) GB

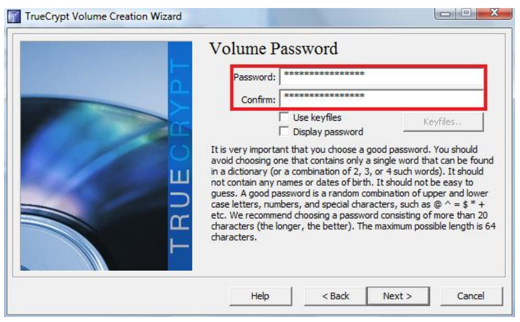
Free space on drive C:\ is 10.08 GB

Please specify the size of the container you want to create.

If you create a dynamic (sparse-file) container, this parameter will specify its maximum possible size.

Note that the minimum possible size of a FAT volume is 275 KB. The minimum possible size of an NTFS volume is 2829 KB.

Figure 5.5: TrueCrypt Step 5



Volume Password

Password: ****************

Confirm: ****************

☐ Use keyfiles

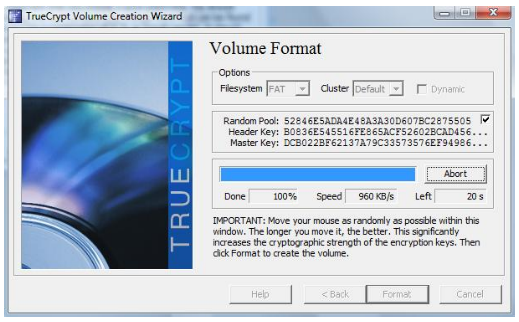
☐ Display password

It is very important that you choose a good password. You should avoid choosing one that contains only a single word that can be found in a dictionary (or a combination of 2, 3, or 4 such words). It should not contain any names or dates of birth. It should not be easy to guess. A good password is a random combination of upper and lower case letters, numbers, and special characters, such as @ ^ = $ * + etc. We recommend choosing a password consisting of more than 20 characters (the longer, the better). The maximum possible length is 64 characters.

Figure 5.6: TrueCrypt Step 6

window (fig 5.7). Once the container is created we must ”mount” it, as if it was an external device, introducing the password (fig 5.8).

When the container is mounted, is possible access it and store data inside as it was an external device. For protecting data stored we only need to dismount the container. If we try to open the container with, for example, notepad we will see string of bytes and characters with apparently no meaning.



Volume Format

Options

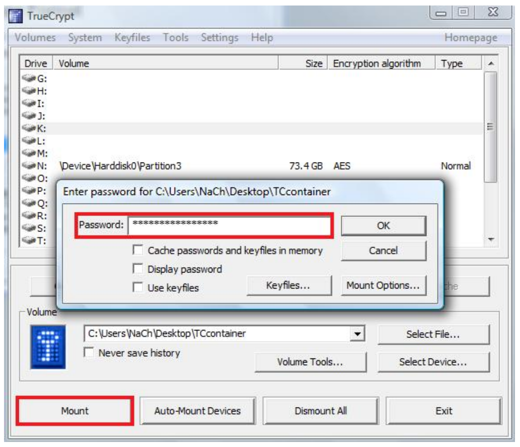
Filesystem FAT Cluster Default ☐ Dynamic

```
Random Pool: 52846E5ADA4E48A3A30D607BC2875505 ✓
 Header Key: B0836E545516FE865ACF52602BCAD456...
 Master Key: DCB022BF62137A79C33573576EF94986...
```

Done 100% Speed 960 KB/s Left 20 s

IMPORTANT: Move your mouse as randomly as possible within this window. The longer you move it, the better. This significantly increases the cryptographic strength of the encryption keys. Then click Format to create the volume.

Figure 5.7: TrueCrypt Step 7



Drive	Volume	Size	Encryption algorithm	Type
G:				
H:				
I:				
J:				
K:				
L:				
M:				
N:	\Device\Harddisk0\Partition3	73.4 GB	AES	Normal
O:				
P:				
Q:				
R:				
S:				
T:				

Enter password for C:\Users\NaCh\Desktop\TCcontainer

Password: ****************

☐ Cache passwords and keyfiles in memory

☐ Display password

☐ Use keyfiles

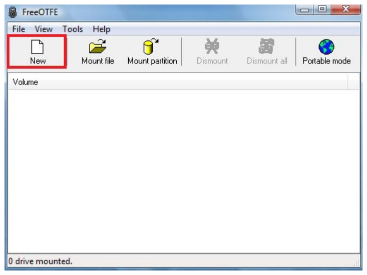
C:\Users\NaCh\Desktop\TCcontainer

☐ Never save history

Figure 5.8: TrueCrypt Step 8

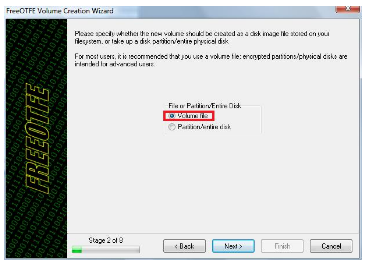
FreeOFTE

The process to create a container with FreeOTFE software is very similar than the described above. First of all we must launch the volume creation Wizard by clicking on the button ”New” of the main window (fig 5.9). Once in the Wizard, after the first informative window, we are going to choose the option ”Volume file” (fig 5.10).



0 drive mounted.

Figure 5.9: FreeOTFE Step 1



Please specify whether the new volume should be created as a disk image file stored on your filesystem, or take up a disk partition/entire physical disk

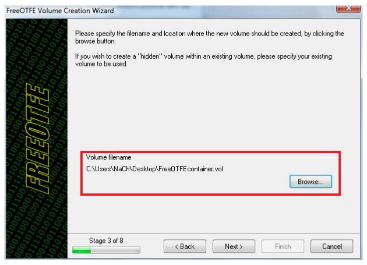
For most users, it is recommended that you use a volume file; encrypted partitions/physical disks are intended for advanced users.

File or Partition/Entire Disk: [✓ Volume file] Partition/entire disk

Stage 2 of 8

Figure 5.10: FreeOTFE Step 2

The following steps will serve to specify the path, the name (fig 5.11) and the size (fig 5.12) for the container. We are going to create it at the same path and with the same size than the container created with TrueCrypt.



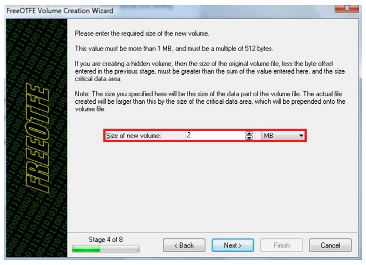
Please specify the filename and location where the new volume should be created, by clicking the browse button.

If you wish to create a "hidden" volume within an existing volume, please specify your existing volume to be used.

Volume filename
C:\Users\NaCh\Desktop\FreeOTFEcontainer.vol

Stage 3 of 8

Figure 5.11: FreeOTFE Step 3



Please enter the required size of the new volume.

This value must be more than 1 MB, and must be a multiple of 512 bytes.

If you are creating a hidden volume, then the size of the original volume file, less the byte offset entered in the previous stage, must be greater than the sum of the value entered here, and the size critical data area.

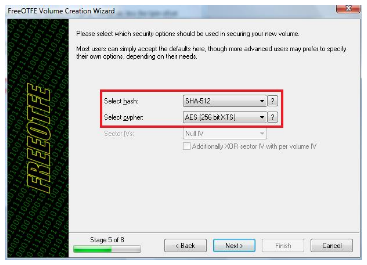
Note: The size you specified here will be the size of the data part of the volume file. The actual file created will be larger than this by the size of the critical data area, which will be prepended onto the volume file.

Size of new volume: 2 MB

Stage 4 of 8

Figure 5.12: FreeOTFE Step 4

Afterwards we are going to choose the encryption mode and the hash algorithm (fig 5.13). Obviously we will select XTS-AES as the encryption mode, and SHA-512 as hash algorithm as we did with TrueCrypt container. The next step will be choose the password for the container (fig 5.14).



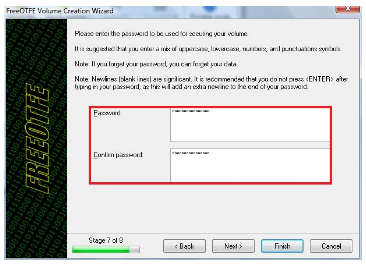
Please select which security options should be used in securing your new volume.

Most users can simply accept the defaults here, though more advanced users may prefer to specify their own options, depending on their needs.

Select hash: SHA-512
Select cypher: AES (256 bit XTS)
Sector IVs: Null IV
Additionally XOR sector IV with per volume IV

Stage 5 of 8

Figure 5.13: FreeOTFE Step 5



Please enter the password to be used for securing your volume.

It is suggested that you enter a mix of uppercase, lowercase, numbers, and punctuations symbols.

Note: If you forget your password, you can forget your data.

Note: Newlines (blank lines) are significant. It is recommended that you do not press <ENTER> after typing in your password, as this will add an extra newline to the end of your password.

Password: ****************
Confirm password: ****************

Stage 7 of 8

Figure 5.14: FreeOTFE Step 6

Finally we only have to finish the Wizard creating the container. To store data inside, we must firstly mount it clicking ”Mount file” and introducing the password (fig **??**). Before we could use the container, we should format it, then the container is ready to be used at the same way as the container created with TrueCrypt.

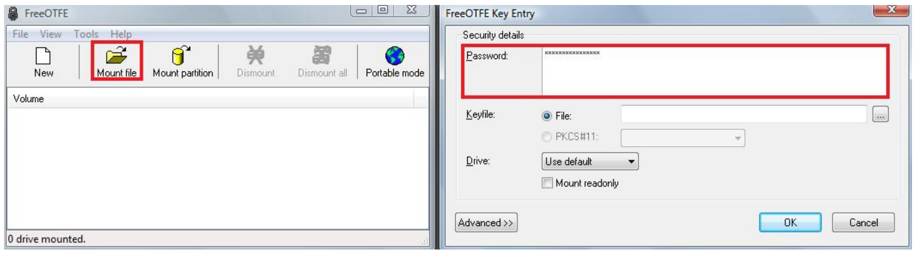


Figure 5.15: FreeOTFE Step 7

Now we stored in both volumes the same file and try to ”open” with a word processor, we will realize that the ”information” we can read is not the same. This is produces because of the randomization introduced by both tools and produce that the same sored data appears as different cipher data. If we try now to open the volume created by TrueCrypt with FreeOTFE, we will be not allowed to do it, because of the metadata that each tool introduce in the container for achieve mounting and decrypting it.

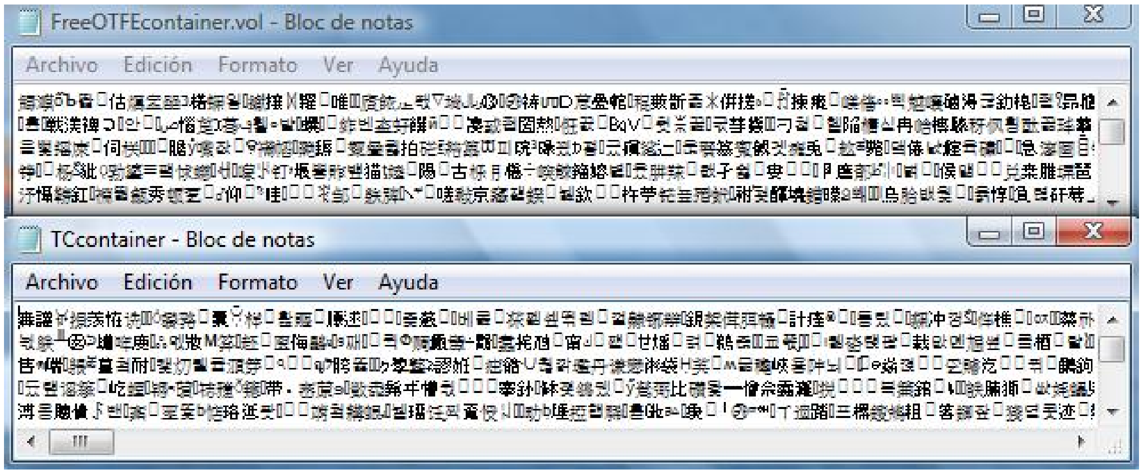

FreeOTFEcontainer.vol - Bloc de notas
Archivo Edición Formato Ver Ayuda
TCcontainer - Bloc de notas
Archivo Edición Formato Ver Ayuda

Figure 5.16: TrueCrypt container vs. FreeOTFE container

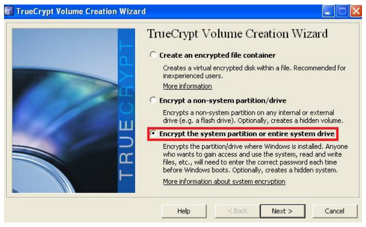
5.1.2 Partition

The process of encrypt a partition is in between volume creation and system encryption process. This process could be conducted by the three encryption tools used: TrueCrypt, FreeOTFE and Diskcryptor. Next section will show the results of encrypting a partition of 73.3 GB in Maachine 1.

5.1.3 System

Another important functionality that present the different encryption software is protect the entire system with XTS-AES encryption mode and authentication. We are going to show the process of encrypting the system in Machine 2 with Windows XP and TrueCrypt. Results for the three different encryption tools will be shown in next section.

TrueCrypt system encryption is started as Container creation process, we must click on ”Create Volume” but this time, we must choose the third option: ”Encrypt the system partition or entire system drive” (fig 5.17). Then we must select the option ”Encrypt the whole drive” (fig 5.18) after selecting a normal type of encryption[1].



TrueCrypt Volume Creation Wizard

( ) Create an encrypted file container

Creates a virtual encrypted disk within a file. Recommended for inexperienced users.

More information

( ) Encrypt a non-system partition/drive

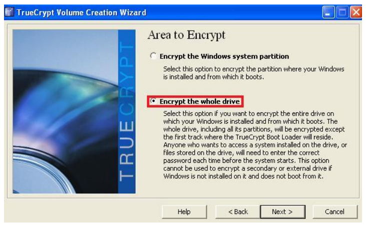
Encrypts a non-system partition on any internal or external drive (e.g. a flash drive). Optionally, creates a hidden volume.

[✓ Encrypt the system partition or entire system drive]

Encrypts the partition/drive where Windows is installed. Anyone who wants to gain access and use the system, read and write files, etc., will need to enter the correct password each time before Windows boots. Optionally, creates a hidden system.

More information about system encryption

Figure 5.17: TrueCrypt system encryption Step 1



Area to Encrypt

( ) Encrypt the Windows system partition

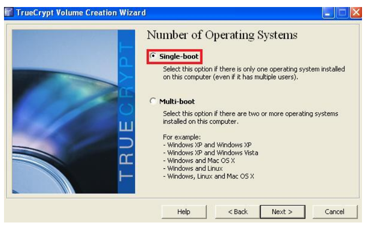
Select this option to encrypt the partition where your Windows is installed and from which it boots.

[✓ Encrypt the whole drive]

Select this option if you want to encrypt the entire drive on which your Windows is installed and from which it boots. The whole drive, including all its partitions, will be encrypted except the first track where the TrueCrypt Boot Loader will reside. Anyone who wants to access a system installed on the drive, or files stored on the drive, will need to enter the correct password each time before the system starts. This option cannot be used to encrypt a secondary or external drive if Windows is not installed on it and does not boot from it.

Figure 5.18: TrueCrypt system encryption Step 2

As in Machine 2 we have only installed Windows XP, we will select the option ”Single-boot” (fig 5.19). Next step will allow us to choose the encryption algorithm we want to use, obviously we will select AES (internally will work in XTS mode). The Hash algorithm in not possible to change it, so it will be RIPEMD-160 (fig 5.20).



Number of Operating Systems

[✓ Single-boot]

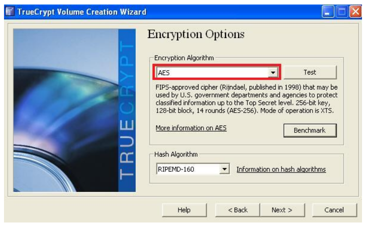
Select this option if there is only one operating system installed on this computer (even if it has multiple users).

( ) Multi-boot

Select this option if there are two or more operating systems installed on this computer.

For example:
- Windows XP and Windows XP
- Windows XP and Windows Vista
- Windows and Mac OS X
- Windows and Linux
- Windows, Linux and Mac OS X

Figure 5.19: TrueCrypt system encryption Step 3



Encryption Options

Encryption Algorithm: AES

FIPS-approved cipher (Rijndael, published in 1998) that may be used by U.S. government departments and agencies to protect classified information up to the Top Secret level. 256-bit key, 128-bit block, 14 rounds (AES-256). Mode of operation is XTS.

More information on AES

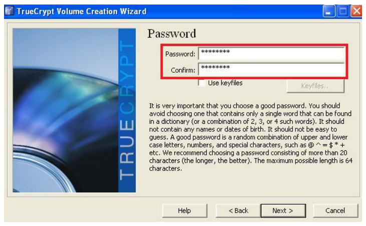
Hash Algorithm: RIPEMD-160

Information on hash algorithms

Figure 5.20: TrueCrypt system encryption Step 4

[1]Hide the system encryption is not an relevant option with the XTS-AES encryption mode.

Afterwards, introduce the password will be necessary to complete next step (fig 5.21). After that we will add uncertainty to the cipher data, for do that we must move the mouse within the Wizard window. When finish it, a part of the created keys will be shown (fig 5.22).

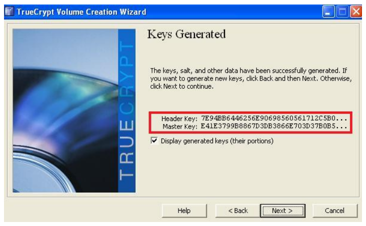


Password

Password: *********
Confirm: *********
Use keyfiles

It is very important that you choose a good password. You should avoid choosing one that contains only a single word that can be found in a dictionary (or a combination of 2, 3, or 4 such words). It should not contain any names or dates of birth. It should not be easy to guess. A good password is a random combination of upper and lower case letters, numbers, and special characters, such as @ ^ = $ * + etc. We recommend choosing a password consisting of more than 20 characters (the longer, the better). The maximum possible length is 64 characters.

Figure 5.21: TrueCrypt system encryption Step 5



Keys Generated

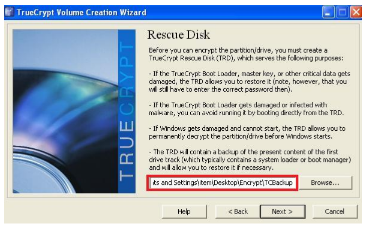
The keys, salt, and other data have been successfully generated. If you want to generate new keys, click Back and then Next. Otherwise, click Next to continue.

Header Key: 7E94BB6446256E90698560561712C5B0...
Master Key: E41E3799B8867D3DB3866E703D37B0B5...

[✓] Display generated keys (their portions)

Figure 5.22: TrueCrypt system encryption Step 6

Now, a TrueCrypt Rescue Disk (TRD) will be created in order to restore the system if some TrueCrypt data necessary to boot the system is damaged or Windows cannot start, it would allow us to decrypt the system and recover it (it contains a backup of the first drive track). We only have to set the path and the name for TRD (fig 5.23) and after that, burn it to a CD. When TrueCrypt detects that we have recorded the rescue disk, will perform a test (fig 5.24) for the necessary components rebooting the system. We will be asked for the password before Windows starts.



Rescue Disk

Before you can encrypt the partition/drive, you must create a TrueCrypt Rescue Disk (TRD), which serves the following purposes:

- If the TrueCrypt Boot Loader, master key, or other critical data gets damaged, the TRD allows you to restore it (note, however, that you will still have to enter the correct password then).

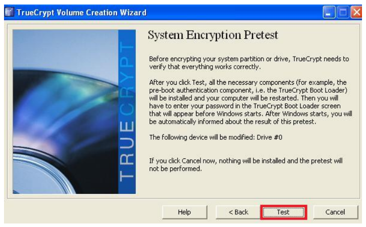
- If the TrueCrypt Boot Loader gets damaged or infected with malware, you can avoid running it by booting directly from the TRD.

- If Windows gets damaged and cannot start, the TRD allows you to permanently decrypt the partition/drive before Windows starts.

- The TRD will contain a backup of the present content of the first drive track (which typically contains a system loader or boot manager) and will allow you to restore it if necessary.

its and Settings\item\Desktop\Encrypt\TCBackup

Figure 5.23: TrueCrypt system encryption Step 7



System Encryption Pretest

Before encrypting your system partition or drive, TrueCrypt needs to verify that everything works correctly.

After you click Test, all the necessary components (for example, the pre-boot authentication component, i.e. the TrueCrypt Boot Loader) will be installed and your computer will be restarted. Then you will have to enter your password in the TrueCrypt Boot Loader screen that will appear before Windows starts. After Windows starts, you will be automatically informed about the result of this pretest.

The following device will be modified: Drive #0

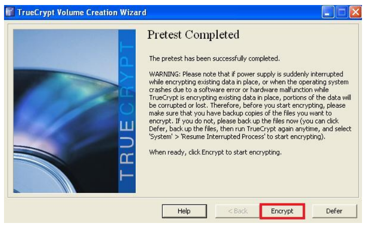
If you click Cancel now, nothing will be installed and the pretest will not be performed.

Figure 5.24: TrueCrypt system encryption Step 8

Finally, after rebooting the system, the encryption could starts by clicking ”Encrypt” (fig 5.25). Once the system encryption process is started, is possible to see the progress and the estimated remaining time (fig 5.26). When TrueCrypt finish the process, a message will appear to inform us. Is it possible to perform the decryption of the system selecting the option ”System/Permanently Decrypt System Partition/Drive”.

5.2 Results

The different results obtained by the various test of each encryption tool will be shown in this section. We will compare all three encryption software in different aspects: volume creation, partition encryption and system encryption.



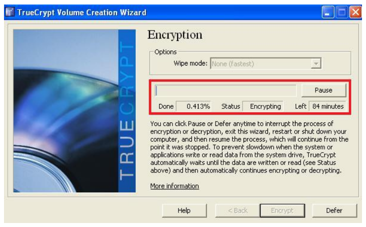
Pretest Completed

The pretest has been successfully completed.

WARNING: Please note that if power supply is suddenly interrupted while encrypting existing data in place, or when the operating system crashes due to a software error or hardware malfunction while TrueCrypt is encrypting existing data in place, portions of the data will be corrupted or lost. Therefore, before you start encrypting, please make sure that you have backup copies of the files you want to encrypt. If you do not, please back up the files now (you can click Defer, back up the files, then run TrueCrypt again anytime, and select 'System' > 'Resume Interrupted Process' to start encrypting).

When ready, click Encrypt to start encrypting.

Figure 5.25: TrueCrypt system encryption Step 9



Encryption

Options

Wipe mode: None (fastest)

Done 0.413% Status Encrypting Left 84 minutes

You can click Pause or Defer anytime to interrupt the process of encryption or decryption, exit this wizard, restart or shut down your computer, and then resume the process, which will continue from the point it was stopped. To prevent slowdown when the system or applications write or read data from the system drive, TrueCrypt automatically waits until the data are written or read (see Status above) and then automatically continues encrypting or decrypting.

Figure 5.26: TrueCrypt system encryption Step 10

5.2.1 Volume creation

Only TrueCrypt and FreeOTFE are able to create containers in order to store a group of files. In table 5.1 is shown the time elapsed during the creation of different size TrueCrypt containers in both Laptop (Machine 1) and Desktop (Machine 2). The times shown in table 5.2 are refered to FreeOTFE encryption software.

Volume Size (MB)	Time Machine 1 (s)	Time Machine 2 (s)
0.5	4.3	0.8
1	4.4	0.8
5	4.4	0.9
10	4.6	1.2
50	6	4.4
100	7.9	7.6
500	21	36.7
1000	35.5	69.2
5000	163.4	261.6
10000	559.8	554

Table 5.1: TrueCrypt Volume Creation

For little size containers, TrueCrypt works faster in Machine 2 with Windows XP, but when the size of the container is bigger than 100 MB it works better in Machine 1 with Windows Vista. An exception occurs when we create a container of 10GB, times elapsed in both machines are similar. One of the characteristics of TrueCrypt when creating containers is that is the program itself is responsible for formating the container. The format is done in FAT.

FreeOTFE works better in Machine 1 when we create container of small size (< 100 MB), however when the size is increased the performance of FreeOTFE in Machine 2 is better. Is necessary clarify the times shown in 5.2 are composed by the addition of the elapsed time of the software and the elapsed time formating the container, the difference between these times lies in the time Windows takes to make the format, as in TrueCrypt containers the format is made in FAT.

Volume Size (MB)	Time Machine 1 (s)	Time Machine 2 (s)
1	1.3	4.5
5	1.8	5
10	3	5.1
50	5.5	6.4
100	25.1	7.9
500	125.3	20
1000	243.8	39
5000	1245.1	168.8
10000	2299.9	331.8

Table 5.2: FreeOTFE Volume Creation

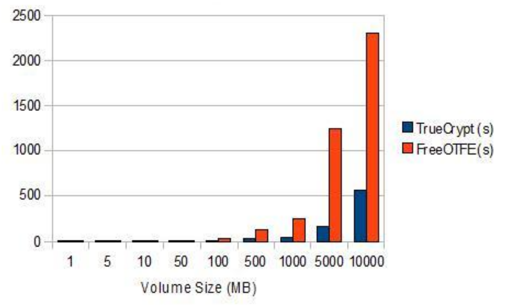


Figure 5.27: TrueCrypt vs. FreeOTFE volume creation in Machine 1

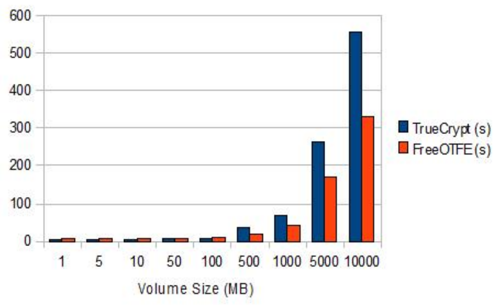


Figure 5.28: TrueCrypt vs. FreeOTFE volume creation in Machine 2

Figures 5.27 and 5.28 show the functionality of TrueCrypt and FreeOTFE in both machines. TrueCrypt presents similar results in both machines. This is not the case of FreeOTFE that presents different performing depending in which machine is executed, also being faster that TrueCrypt in Machine 2 with Windows XP.

5.2.2 Partition encryption

We have tested the different encryption software by encrypting a partition of 73.3 GB in Machine 1. DiskCryptor allows us to encrypt and subsequently decrypt the partition. Both TrueCrypt and FreeOTFE encrypt the partition and convert it in a big container.

FreeOTFE	TrueCrypt	DiskCryptor
157.32	130.45	115

Table 5.3: Partition Encryption (minutes elapsed)

Encryption	Decryption
115	131

Table 5.4: DiskCryptor: Partition encryption and decryption time (minutes elapsed)

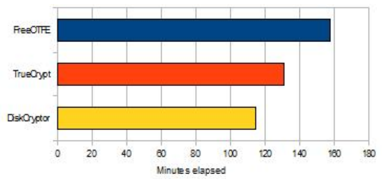


Figure 5.29: Partition Encryption

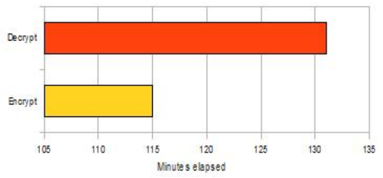


Figure 5.30: DiskCryptor: Partition encryption and decryption time

Table 5.3 and figure 5.29 show the minutes elapsed by each encryption tool to perform the partition encryption. DiskCryptor spend less time encrypting and also it could undone the encryption (table 5.4 and fig 5.30).

5.2.3 System encryption

Finally, we encrypted the entire system in Machine 2 using the different encryption tools.

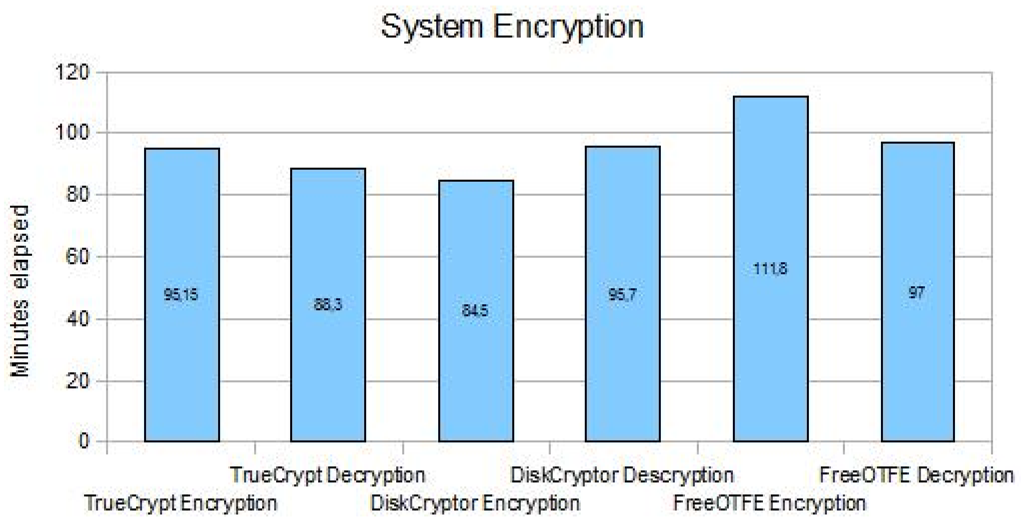


Figure 5.31: System encryption

DiskCryptor is the faster encryption software and the unique that spends more time decrypting than encrypting. All three tools ask for the password we entered during the encryption config before booting the system.

Chapter 6

Conclusion

Throughout this Thesis we have conducted a comprehensive analysis of the new standard published by the IEEE and developed by SISWG: *1619.1 Standard for Authenticated Encryption with Length Expansion for Storage Devices*. We have seen different aspects related such as AES encryption algorithm in which is based on, the implementation of new techniques to provide greater security, comments and opinions outside the SISWG trying to ascertain its functionality and exposing any weaknesses, and finally its application commercial and professional level.

The AES cipher is a secure system in which there is no known effective attack that can break its security and to reveal the key, XTS operation mode provides greater security the standard due to the use of AES block cipher for encrypting both plaintext and tweak values, that add uncertainty in order to avoid repeated ciphertext when trying to encrypt the same plaintext. Moreover Ciphertext Stealing technique used to encrypt data units that are not fully divisible by 128-blocks is functional and safe, and NIST implements it in some of its systems.

The use of two keys in the XTS encryption mode, one for each AES block cipher, and even adds greater security to greatly diminish the chances of decrypting if they are unknown. XTS is based on XEX mode developed by Rogaway, proved that using the same key for two different objectives is enough secure, XTS perfects it varying the key of each of these goals.

Several reviews have been submitted to NIST after the IEEE published the standard and SISWG has made a public consultation to gather comments. Most of these comments are related to the XTS-AES algorithm itself: issues such as security, the use of two keys or Ciphertext Stealing technique have been the most required and subsequently rationalized by SISWG. Other aspects such the industry support for the new algorithm and its future applications have also been discussed.

The support of the industry is large and there are many developers, both software and hardware, which have shown interest and have implemented XTS-AES encryption mode in their products. Throughout this Thesis we have named some of them and we have analyzed different tools that implement encryption XTS-AES: TrueCrypt, FreeOTFE and DiskCryptor, as well as the IP cores that perform it also.

The future implementation of the XTS-AES encryption mode will be comprehensive, not only by the amount of industrial and commercial products that support and implement it, but because of the development work that continue the IEEE and the SISWG related with the security of the stored data:

- 1619.2 Standard for Wide-Block Encryption for Shared Storage Media
- 1619.3 Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data

Finally we must emphasize that the XTS-AES encryption mode is a secure and reliable system for safely storing sensitive information and the different variations that characterize it will not add undue complexity. The industrial support is broad and with which the application future is hopeful, while pending the standardization by NIST.

Glossary

α	A primitive element of $GF(2^{128})$ that corresponds to polynomial x
$\bullet$	Assignment of a value to a variable
$\lfloor x \rfloor$	Floor of x
$\oplus$	Bit-wise exclusive-OR operation
$\otimes$	Modular multiplication of two polynomials over the binary field $GF(2)$, modulo $x^{128} + x^7 + x^2 + x + 1$
Affine Transformation	A transformation consisting of multiplication by a matrix followed by the addition of a vector.
Array	An enumerated collection of identical entities (e.g., an array of bytes).
Bit	A binary digit having a value of 0 or 1.
Block	Sequence of binary bits that comprise the input, output, State, and Round Key. The length of a sequence is the number of bits it contains. Blocks are also interpreted as arrays of bytes.
Blowfish	Is a symmetric block cipher that can be used as a drop-in replacement for DES or TDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.
Byte	A group of eight bits that is treated either as a single entity or as an array of 8 individual bits.
CAST5	Is a symmetric block cipher with a block-size of 8 bytes and a variable key-size of up to 128 bits.
Cipher	Series of transformations that converts plaintext to ciphertext using the Cipher Key.

Cipher Key	Secret, cryptographic key that is used by the Key Expansion routine to generate a set of Round Keys; can be pictured as a rectangular array of bytes, having four rows and Nk columns.
Ciphertext	Data output from the Cipher or input to the Inverse Cipher.
Data unit	Within IEEE Std 1619, 128 or more bits of data within a key scope. The first data unit in a key scope starts with the first bit of the key scope; each subsequent data unit starts with the bit after the end of the previous data unit. Data units within a key scope are of equal sizes. A data unit does not necessarily correspond to a physical or logical block on the storage device.
DES-X	Is a variant on the DES (Data Encryption Standard) block cipher intended to increase the complexity of a brute force attack using a technique called key whitening.
Hash function	A function that maps keys to integers, usually to get an even distribution on a smaller set of values.
Inverse Cipher	Series of transformations that converts ciphertext to plaintext using the Cipher Key.
IP core	An IP (intellectual property) core is a block of logic or data that is used in making a field programmable gate array (FPGA) or application-specific integrated circuit (ASIC) for a product.
Key Expansion	Routine used to generate a series of Round Keys from the Cipher Key.
Key scope	Data encrypted by a particular key, divided into equal-sized data units. The key scope is identified by three non-negative integers: tweak value corresponding to the first data unit, the data unit size, and the length of the data.(See [13])
MD5	Message-Digest algorithm 5. Is a widely used cryptographic hash function with a 128-bit hash value.

Plaintext	Data input to the Cipher or output from the Inverse Cipher.
Rijndael	Cryptographic algorithm specified in this Advanced Encryption Standard (AES).
RIPEMD-160	Is a 160-bit hash function. Is intended to be a replacement of MD4, MD5 and RIPEMD hash functions.
Round Key	Round keys are values derived from the Cipher Key using the Key Expansion routine; they are applied to the State in the Cipher and Inverse Cipher.
S-box	Non-linear substitution table used in several byte substitution transformations and in the Key Expansion routine to perform a one-for-one substitution of a byte value.
Serpent	Is a symmetric key block cipher which was a finalist in the Advanced Encryption Standard contest, where it came second to Rijndael. Has a block size of 128 bits and supports a key size of 128, 192 or 256 bits.
SHA-512	Secure Hash Algorithm. The SHA hash functions are a set of cryptographic hash functions. SHA-512 belongs to the SHA-2 family, all the mebers of the family use an identical algorithm with a variable digest size.
State	Intermediate Cipher result that can be pictured as a rectangular array of bytes, having four rows and Nb columns.
Tiger	Is a cryptographic hash function designed for efficiency on 64-bit platforms. The size of a Tiger hash value is 192 bits.
Triple DES	Is the common name for the Triple Data Encryption Algorithm (TDEA) block cipher, it applies the Data Encryption Standard (DES) cipher algorithm three times to each data block.

Twofish	Is a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. It was one of the five finalists of the Advanced Encryption Standard contest, but was not selected for standardisation. Twofish is related to the earlier block cipher Blowfish.
Word	A group of 32 bits that is treated either as a single entity or as an array of 4 bytes.

Bibliography

[1] Mihir Bellare and Phillip Rogaway, “Introduction to Modern Cryptography,” 2005.

[2] Jose Luis Tabara, “Breve historia de la criptografía,” 2007.

[3] Dr. A. Ray Miner, “The Cryptographic Mothema tics of Enigma,” *NSA/CSS*.

[4] RSA Laboratories, “What is a block cipher?.”

[5] Laurent Haan, “Advanced Encryption Standard (AES).”

[6] FIPS, “Advanced Encryption Standard,” *NIST*, no. 197, 2001.

[7] Reach Information, “Disk Encryption Theory: Synonyms, Definition and Meaning about disk encryption theory from Reach Information.”

[8] Bruce Schneider, *Applied Cryptography*. John Wiley and Sons, 1996.

[9] Weisstein, Eric W, “Finite Fields From MathWorld–A Wolfram Web Resource.”

[10] Moses Liskov, Ronald L. Rivest and David Wagner, “Tweakable Block Ciphers,” 2003.

[11] P. Rogaway, M. Bellare, J. Black, and T. Krovetz, “OCB: a block-cipher mode of operation for efficient authenticated encryptiona,” *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pp. 196–205, 2001.

[12] Phillip Rogaway, “Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC,” 2003.

[13] IEEE Computer Society, “IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices,” *IEEE*, no. 1619, 2007.

[14] NIST, “Public Comments on the XTS-AES Mode.”

[15] Moses Liskov and Kazuhiko Minematsu, “Comments on XTS-AES,” 2008.

[16] Matthew V. Ball, Sun Microsystems, Chair of IEEE Security in Storage Working Group (P1619), “Public Comments on the XTS-AES Mode.”

[17] Willett, Michael, Seagate Technology, “Comments provided to NIST in response to: Request for Public Comment on XTS/AES.”

[18] IP Cores, Inc., “XTS3 Family of Cores,” 2007.

[19] IP Cores, Inc., “XTS2 Family of Cores,” 2007.