Connector, investor, futurist, geek, software developer, innovator, sportsman, libertarian, business enabler, cosmopolitan, autodidact, funny finch, tech evangelist,
purist, agnostic, Kärnten fan, foodie, artist, globetrotter, social liberal but fiscal conservative, Schöngeist... elegantiorum litterarum amans oder studiosus...

This is the website of Markus Gattol. It is composed, driven and secured/encrypted exclusively by Open Source Software. The speciality of this website is that it is seamlessly integrating into my daily working environment (Python + MongoDB + Linux + SSH + GIT + ZeroMQ) which therefore means it becomes a fully fledged and automatized publishing and communication platform. It will be under construction until 2014.

Open Source / Free Software, because freedom is in everyone's language...
Frihed Svoboda Libertà Vrijheid เสรีภาพ Liberté Freiheit Cệu̯-iù Ελευθερία Свобода חרות Bebas Libertada 自由

Website Sections

Home
FAQs
About Me

Tweets by @markusgattol

Block-layer Encryption

Status: This page is considered finished. Changes/reviews might happen.
Last changed: Saturday 2015-01-10 18:31 UTC

Abstract:

Block-layer encryption, also known as "whole disk encryption", "on-disk encryption" or "full-disk encryption" is a kind of disk encryption software or hardware which encrypts every bit of data that goes onto a disk, disk partition or disk volume of some sort (LUN, RAID Volume, ordinary disk, etc.). The term "full-disk/on-disk encryption" is often used to signify that everything on a disk is encrypted, including the programs that can encrypt bootable operating system partitions. Block-layer encryption is in contrast to filesystem-level encryption, which is a form of disk encryption where individual files or directories are encrypted by the filesystem itself. The enterprise-class block-layer encryption for Linux goes by the name dm-crypt. There is also an extensions to it called LUKS (Linux Unified Key Setup) which enables us to do fancy things like key management for example. dm-crypt and LUKS, both are free-software working together in order to provide data encryption on storage media thus allowing that what is a secret stays a secret. dm-crypt is a device-mapper and part of the Linux operating system kernel. LUKS is a hard disk encryption specification, represented by cryptsetup, its actual implementation. This page discusses the motivation behind encryption and provides a guide in order to setup and manage encrypted storage media.

Table of Contents

Motivation behind Encryption
- Justifying Encryption
- Encryption is a Necessity and a Godsend

Techniques
- Booting
- Stack

Theory
Theory and Practice
- DM-Tables
- Password/Key Files and Keys
- Salt, Stretching
- Destroy Data, IT Forensics
- Ciphers, Chaining Mode, IV, Watermark
- Performance
- Ciphers and Modes of Operation

Setup and Manage
- Prepare
- Correct Device
- External/Internal Hard Disk Drive
- Logical Volume Manager
- Hardware RAID
- Solid State Drive
- USB Stick
- Swap Partition
- Automatize

Miscellaneous

Note: As mentioned, the other type of encryption next to block-layer encryption is filesystem-level encryption. Following that link, one will also read about some thoughts when either type (block-layer or filesystem-level encryption) might be used. As usual, there is no *better in general* — it depends on the use case and the environment we are confronted with in a particular situation.

Motivation behind Encryption

Modern businesses, human rights groups, political minorities, federal agencies, individuals, the military... they all have one thing in common — they want to sustain their integrity, self-determination (links to German site) or maybe just their way of living and/or working by not letting anybody know anything about them. **With all the data nowadays stored on computers and the information technologies humans use, all those gadgets and devices have more or less become peoples second brain — certainly, nobody wants anyone else reading his minds**.

If we look a little closer, we might find very good and precise reasons why it should be commonsense to encrypt data and why it has to be accepted by anybody. Therefore, we need to take a look at our political as well as social environment and some terms.

Encryption

The reversible transformation of data from the original (the plaintext) to a difficult-to-interpret format (the ciphertext) as a mechanism for protecting its confidentiality, integrity and sometimes its authenticity. Encryption uses an encryption algorithm and one or more encryption keys. *(The Free Computing Dictionary)*
In cryptography, encryption is the process of transforming information (referred to as plaintext) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as ciphertext). In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. *software for encryption* can typically also perform decryption), to make the encrypted information readable again (i.e. to make it unencrypted). *(Wikipedia)*

Justifying Encryption

Let us introduce two basic terms. Secrecy and privacy. Before you go, look them up somewhere try to answer to yourself, offhanded, what their meaning is. Mostly, when asked instantly, people get it wrong or cannot answer it completely. Another common term among geeks[1] but also ordinary people when it comes to computer and modern communication related security is paranoia[2]. There are more but those three are the ones that suffice in justifying why a group[3] of humans would want to use encryption.

Secrecy

Secrecy is the practice of sharing information among a group of people, which can be as small as one person, while hiding it from others. That which is kept hidden is known as the secret. Secrecy is often controversial. Many people claim that, at least in some situations, it is better for everyone if everyone knows all the facts — there should be no secrets. *(Wikipedia)*

1. The quality or condition of being secret or hidden; concealment.
2. The ability or habit of keeping secrets; closeness. *(The Free Dictionary)*

Privacy

Privacy has no definite boundaries and it has different meanings for different people. It is the ability of an individual or group to keep their lives and personal affairs out of public view, or to control the flow of information about themselves. The right against unsanctioned invasion of privacy by the government, corporations or individuals is part of many countries laws, and in some cases, constitutions or privacy laws. *(Wikipedia)*

1. The quality or condition of being secluded from the presence or view of others.
2. The state of being free from unsanctioned intrusion: a person's right to privacy.
3. The state of being concealed; secrecy. *(The Free Dictionary)*

Paranoia

Paranoia is a disturbed thought process characterized by excessive anxiety or fear, often to the point of irrationality and delusion. Paranoid thinking typically includes persecutory beliefs concerning a perceived threat. Paranoia is distinct from phobia, which is more descriptive of an irrational and persistent fear, usually unfounded, of certain situations, objects, animals, activities, or social settings. By contrast, a person suffering paranoia or paranoid delusions tends more to blame or fear others for supposedly intentional actions that somehow affect the afflicted individual. *(Wikipedia)*

1. No chance I'm gonna give a definition.. What if someone is watching:? :-o *(The Urban Dictionary)*
2. A psychotic disorder characterized by delusions of persecution with or without grandeur, often strenuously defended with apparent logic and reason.
3. Extreme, irrational distrust of others. *(The Free Dictionary)*

Now that we got a notion about those three terms I am going to introduce another two essential terms — human rights and self-determination — which, by including the above, are just right for deriving the rationale for *why encryption is a good thing* and by doing so providing an answer to the question of what about *the motivation behind encryption* is.

Human Rights

The basic rights and freedoms to which all humans are entitled, often held to include the right to life and liberty, freedom of thought and expression, and equality before the law. *(Wikipedia)*

And how does all this fit together nicely in order not just only to justify the effort but much more importantly make a stand and claim the right to use encryption for anybody at any times under any circumstances? Well, my point of view is it is all about

Self-determination

1. Determination of one's own fate or course of action without compulsion; free will.
2. Freedom of the people of a given area to determine their own political status; independence.
   (The Free Dictionary)

We need to look at this term from two different points of view. The first one being the one that tells about the integrity and self-determination focused towards whole nations. The second one, and as I find the more suitable/important point of view with regards to encryption is the one focusing on self-determination on a per individual basis — again, a group of people consisting of at least one person. Self-determination is a central principle of human rights... my understanding of it is:

Humans should be able to realize their own ideas, dreams and desires without being disabled or hassled from the outside. Acting, learning and living in a self-determined manner can be seen as flags of an adult human being that acts in free will, autonomous and responsible not just for himself but also for his environment. That in mind, self-determination is by far not to be confused with arbitrariness and self-righteousness since such behavior would harm a human beings environment and in turn penetrate somebody else "space", decreasing his degree of self-determination.

If you are able to live without breaking this basic principle you might unfortunately be part of a minority. The next section is intended to summarize and finally provide a reasoning why block-layer data encryption is a good thing.

Encryption is a Necessity and a Godsend

Rationale 1: Now, that we've entered times where governments, arbitrary institutions or corporations as well as individuals[4] think they can act against human rights and do whatever they desire in the name of something stupid like the war on terror[5][6] for example, usual folks like you and me see themselves forced to protect their self-determination on their own. To my understanding it is quite commonsense that, in order to be declared a target, somebody needs to know something about you — if he does not (or if he maybe even does not know about your existence which is even better) he can hardly harm you.

Statement 1

Individuals can protect themselves by honoring their privacy and thereby sustain their degree of self-determination. This, of course, is perfectly right. Keeping one's privacy is a human right (Article 12, United Nations - Human Rights Charta). Individuals have a right for privacy and therefore a right to use block-layer encryption to retain their privacy. If you are using block-layer encryption and somebody pressures you to reveal what you consider is part of your privacy, he is acting against human rights and can be seen as a criminal no matter what he is/represents does or says. In this case, behind you are all the United Nations — behind the self-elected ruler nothing more than just a pervert mind. So, if you are an individual and you doesn't use block-layer encryption yet, then do not hesitate and start using it.

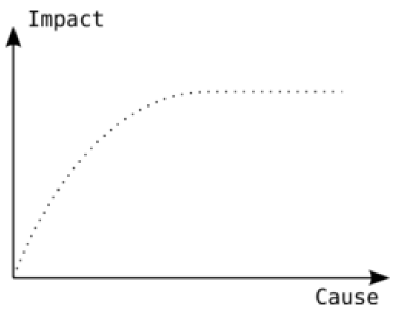
Rationale 2: You are no human being. You are what humans call a *business*. Humans put a lot of effort into you, sometimes they love you, sometimes they hate you but no matter what... they need you and you need them. The reason for your existence is founded and based on the existence of a bunch of humans that march into the same direction by sharing a common mindset and by talking efforts that lead to projected outcomes. You are, more then ever before, subject of a whole variety of substantial threats that may cause damage to you. You, can catch a disease and become ill for example. If you're ill, you loose momentum and so do all the humans representing you as a single unit to the outside world. Nowadays, that you are so much reliant on information and the way you deal with it, if you make mistakes in this regaras, you can become very ill. Not having the correct information at the right times in the right places, pretty much equals problems if not dead.

Statement 2

Modern businesses are real addictives. To the same degree a junkie needs his jab, our economies have an ungovernable need for raw materials, energy and information. This paper is about information, therefore I will not talk about things like oil or electrical energy but about how to secure the existence of information in order to keep the economy going (at least that little part of the economy that you as a human being feel affiliated with). It is all about control. *Who, at what times to what conditions decides who knows what and how much does it cost him to know it after all.* If you can't secure information others feel the need to know about[7], you can't secure your business, can't secure your colleagues nor your own existence. *The one who controls information in the afore mentioned way has the powers, the one who doesn't is the applicant.* Block-layer

encryption is the key to put you into control of all the information that circulates within your business and therefore is worth plain money.
If you are customer, you should ask a company what measures they take in order to avoid data breach with drastic consequences for you — you have certainly already heard about credit card information and the like that *has gone missing*. In fact, gone missing would not be a problem for customers but what mostly happens is that your personal data has fallen into hands of criminals or even worse. That is of course happening much more often than the press reports since companies do what ever they can to prevent such information from becoming public — shares would fall and no marketing division could repair such damage done to a companies reputation. Next, think about you going to a doctor, telling him all the things you would not speak loud about in public. How much would you pay someone to not reveal that you have a disease? This could be reason you are getting fired from your job if anybody gets information about you having some disease (also if not crucial nor infective) or it becomes public your psychologist thinks that you are a real weirdo. As companies to which you are a customer, also your doctor should use encryption in order to protect you — ask them if they do! If not, find one who cares about you as a customer paying them fair money.
A breach of such a magnitude can happen very easily. Think about the random employee from some random business or random medical office and then his car gets stolen. Hm... what has somebodies car do with your personal data? Well, folks carry around laptops, USB sticks etc. If they drive cars, and if it gets stolen your secrets are no secrets anymore. Would the data on the laptops hard disk have been encrypted, nothing of the information stored onto that device could cause harm to anybody since nobody could *read*[8] the information stored on that particular hard disk. This is just one of a thousand examples but the principle is always the same. The only protection against data breach is block-layer encryption and not hiding away your computer in the basement or such nonsense.

Rationale 3: To my understanding and from my point of view towards mankind, the third big argument for encryption is balance and saturation. Before we take a look into an encyclopedia let us ask what physics can tell us about saturation. There is a hard believe among physicist that only physical systems with self protection exist. In simple words that means, that there is no linearity in magnitude between a cause and its impact. From a certain point onwards, if the cause grows bigger, the related impact does not grow anymore or at least at such a tiny magnitude that, practically speaking, there can be no more relationship observed between a cause and an impact. All that is shown by the graph below.



Relationship between cause and impact

A simple example of the above, everybody should be able to realize, is the relation between a humans will (cause) and body movement (impact). One can move his arms around at different speeds — his will can control the speed. He can move his arms not at all, very slow, a little faster, pretty fast and as fast as he can — in fact, everything in between not at all and as fast as he can. Definitely, there is the relationship between his mind (cause) and the speed of his arms moving (impact). The point where one cannot move his arms faster even if he wants to do so, is where the relationship between cause and impact seems to vanish — the curve turns into an horizontal line.

This is self protection since humans are able to imagine moving their arms around with the speed of light but they would simply die by doing so since centrifugal force would be way bigger than the force that keeps arms connected to a humans torso.

Ascertainment 1: We assume, existing systems of any kind know some sort of saturation otherwise they would have already vanished.

Ascertainment 2: In the metaphysical or conceptual sense, balance is used to mean a point between two opposite forces that is desirable over purely one state or the other, such as a balance between the metaphysical Law and Chaos — law by itself being overly controlling, chaos being overly unmanageable, balance being the point that minimizes the negatives of both. *(Wikipedia)*

So, what has all the saturation and balance stuff to do with block-layer encryption? It helps to keep the balance between rulers and the rest of our species. Block-layer encryption maybe just a tiny but nonetheless an important counterbalance that helps our species keeping balance between chaos[9] and total slavery.

Statement 3

> At any times, there have been, currently there are, and most probably there will be rulers. A minority decides what is best[10] for the majority of our species. Unfortunately, we do not live in a system based on the idea of anarcho-capitalism but on what rulers call democracy — clearly, nowadays the system we are living in may be labeled *democracy* but it certainly isn't that kind of democracy that the good old Greeks invented and did well with. Groups of people can secure there being and daily tasks, political and social as well as economical ideas and efforts by using block-layer encryption and thus preventing men with power to get what men with powers are always seeking for — more power. Reread ascertainment number two. In short, no balance, no human race...

Now, after a philosophical journey which some of you might find amusing, others I might have confused and a portion of you is going to throw stones and whiskey bottles (hopefully you emptied them before) at court buildings (please don't do that, better start using block-layer encryption) we should probably get to grips with block-layer encryption. Just a last side blow: If nonetheless you find I am a weirdo and you're with the sane folks, you are either a political leader, you are listed among the richest (which, by the way, couldn't even manage to stop mass starvation, wars nor global climate crisis and the extinction of tons of species etc.) or just not informed enough.

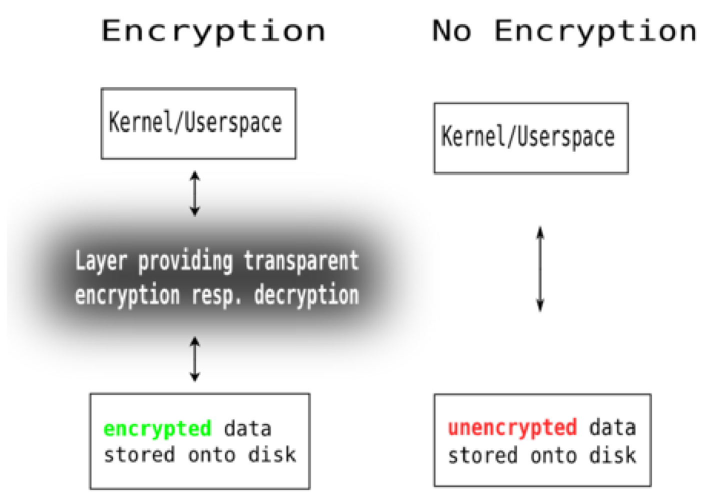
However, do not take my philosophical thoughts to serious since I am a) no professional philosopher, b) not a professional politician (so it's easy to grouch a bit towards those folks) and c) I am not a *Listen to me since I am the one and only messiah that can bring you <whatever_you_desire>* guy. In fact I am just a free spirit guy who likes to scratch on peoples point of view every now and then...

Techniques

This section shows some techniques used to encrypt data...

Booting

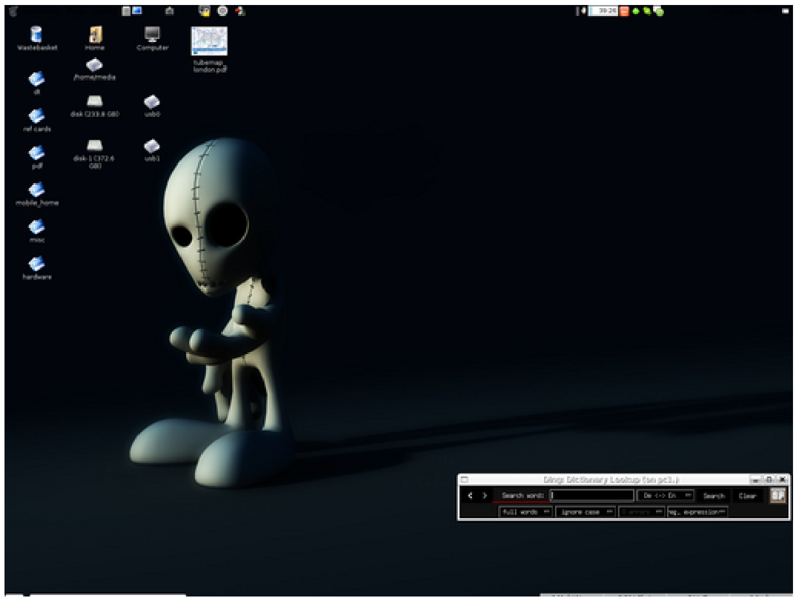
When using block-layer encryption, when the computer boots, the computer asks for a password and/or a keyfile e.g. stored on an USB stick in order to continue booting i.e. loading the operating system. If none or not the correct password is provided, nothing happens. The computer simply refuses to continue with booting thus the data stored on the disk cannot be *seen* — all information stored on the disk is hidden until somebody provides the correct password/keyfile or both in case of the more paranoid. It is very important that the boot password is complex[11] as if the password is simple then the protection of the block-layer encryption is nullified by simple password. A weak passwords always trumps strong security methods.


Encryption
No Encryption
Kernel/Userspace
Kernel/Userspace
Layer providing transparent encryption resp. decryption
encrypted data stored onto disk
unencrypted data stored onto disk

Encryption, an intermediate layer

As the sketch above shows, on the left side a system that uses encryption e.g. dm-crypt and LUKS and on the right side a *normal* system without encryption. As you can see, the important thing here is that on the left data stored on the computers hard disk drive is encrypted and on the right it is not. The box called Layer providing encryption resp. decryption at the left is where all the magic happens, to whom you provide your password and/or keyfile and which in turn makes any data stored on the disk only available to the person that knows either the correct password and/or has the correct keyfile. Ideally that would be you and only you. In cases, for whatever reason, someone else but you sits in front of your computer, he cannot *read* your data even if he takes the disk out of your computer and connects it to his own or any other device.

If you do not know about kernel or userspace anyway yet, do not worry, there is no need to in order to create and use an encrypted system — all you need to know is what you want is what the left column shows. If you are using an operating system like Windows, Linux or just love your overpriced Apple thingy and you are not a programmer or the like you are probably looking at something like the image[12] below.



My current desktop - GNOME on DebianGNU/Linux

Upper left corner of my desktop

Upper left corner of my desktop

There, on the desktop, you have all your icons that you use to start your web-browser, email application, music application etc. — if you do that and your system uses block-layer encryption then

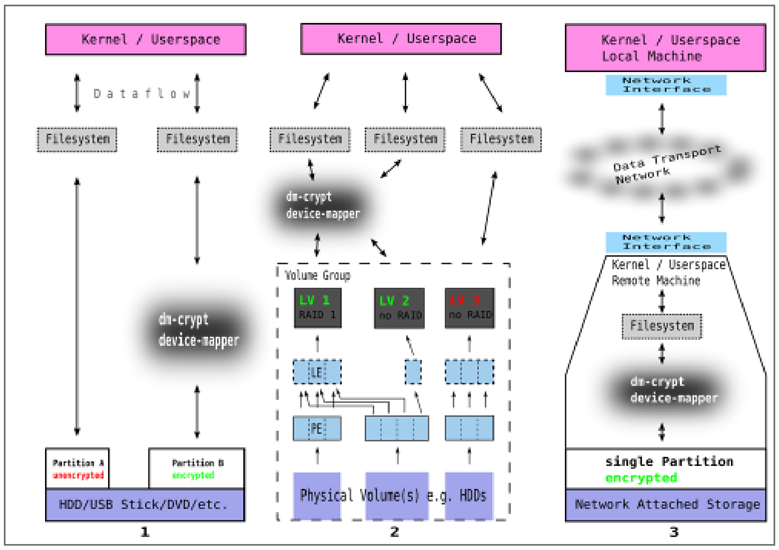
- For you it makes no difference — everything works as before — this is know as transparency.
- All your data e.g. emails, photos, music, videos, etc. are stored onto the encrypted disk — no one but the person with the correct password and/or keyfile can unlock the disk.

Stack

There is a layered logic included when using block-layer encryption with dm-crypt. There is a whole variety how one might use block-layer encryption with dm-crypt. One can just encrypt his USB stick, he can encrypt his external USB disk or the internal disk in his computer. The hardware can be as small as an embedded device e.g. some SD card storage on a submarine robot or as huge as a rack-mountable multiprocessor, multicore systems that is installed at a DC (Data Center) which is controlled by staff located in some NOC (Network Operations Center). Our personal computer situation is probably somewhere in between those cases.

One could encrypt a single disk as a whole or just one or more partitions on a single disk. With servers, it is common to use RAID (Redundant Array of Independent Drives (or Disks)) arrays for reasons of increased I/O (Input/Output) and redundancy as well as to ease administration. Those can be in soft- as well as hardware (i.e. a hardware RAID). Adding to all the afore mentioned is the ability to use things like LVM (Logical Volume Manager), EVMS (Enterprise Volume Management System)[13] and the like and a variety of filesystems.

A selection of three common examples using dm-crypt:



Terms

Before we go into detail with the three examples above, we need to know a little about technical jargon related to our case.

Hard Disk Drive, Disk, Hard Disk (HDD)

The primary computer storage medium, which is made of one or more aluminum or glass platters, coated with a ferromagnetic material. Most hard disks are *fixed disks,* which have platters that reside permanently in the drive. Removable disks are encased in plug-in cartridges, allowing data to be taken out of the drive for storage or for transfer to another party. Before high-speed connections were common on the Internet, removable SyQuest, Jaz and Zip disks were routinely sent through the post office. More after the jump ... *(The Free Computing Dictionary)*

USB Stick

USB flash drives are NAND-type flash memory data storage devices integrated with a USB (Universal Serial Bus) interface. They are typically small, lightweight, removable and rewritable. As of April 2007, memory capacities for USB Flash Drives currently are sold from 32 megabytes up to 64 gigabytes. Capacity is limited only by current flash memory densities, although cost per megabyte may increase rapidly at higher capacities due to the expensive components. More after the jump... *(Wikipedia)*

Partition

A hard disk partition can span a single whole disk, providing all the physically available space with one partition. Another possibility is to divide the physical space into more than one partition. The third possible case is, that some space of an hard disk drive is not made into a partition at all, thus being unused.

File System (FS)

In computing, a file system (often also written as filesystem) is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a data storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol (e.g., NFS, SMB, or 9P clients), or they may be virtual and exist only as an access method for virtual data (e.g., procfs). *(Wikipedia)*

Computer Network resp. Data Transport Network

A network of data processing nodes (e.g. computer, GPS unit, mobile phone, automatic pet feed station, heating, etc.) that are interconnected for the purpose of data communication. A computer connected to a non-computing device (e.g. Network-attached storage, a printer via an Ethernet link, etc.) may also represent a computer network.
Although I named the storage NAS (Network-attached storage), this can be any kind of storage where data stored on a remote device is accessed by another device and therefore has to travel through a computer network.

Device Mapper

Go here here and return afterwards.

Physical Volume (PV)

A physical volume (PV) is represented by a hard disk partition or some hard disk partitions as a whole forming a software or hardware RAID (Redundancy Arrays of Independent Disks) device that can be used the same way as any *normal* hard disk partition. On big server systems for example, it is very common to create PVs consisting of a bunch of hard disk drives connected to a RAID HBA (Host Bus Adapter) hardware controller card thus providing a maximum of I/O (Input/Output) and redundancy to the connected computer system. As a side effect TCO (Total Cost of Ownership) is reduced since it eases and allows to speed-up administrative tasks regarding such a computer system.

Physical Extend (PE)

Is a chunk of storage space located onto a physical volume (PV). Traditional volume managers (such as those in HP-UX and Linux) have PEs of a uniform size; others have variably-sized PEs that can be split and merged at will.

Logical Extend (LE)

Whereas a PE is explicitly assigned to a particular PV, a LE can be assigned to one or more PEs (it is not directly bound to a PV but has at least one intermediate PE between itself and the PV(s) and that in turn if data is written to a LE that has assigned more than one PEs on different PVs, data is written to all the PVs that provide PEs to that particular LE. In case, a LE is represented by two or more PEs onto different PVs, some sort of RAID is accomplished.
So, as you can see, LVM is also capable of providing some sorts of redundancy by spreading data across more than one PV. Also, PEs and LEs have (as far as I know) always the same size — therefore the terrific mapping between them.

Logical Volume (LV)

A multitude of concatenated LEs forms a LV. If for example the chunk size of a LE is 2 MiB, taking 2000 LEs forms a LV of size 4 GiB. These LVs are virtual devices that behave just like hard disk partitions — mountable file systems can be created on them. Each LV represents addressable consecutive space of a block storage.

Volume Group (VG)

A VG is a grouping of some PVs (together with all the PEs/LEs/LVs contained on those PVs). Both PVs and LVs cannot span multiple VGs, they always belong to a single VG. A VG is the main unit of administrative actions. One can online/offline[14] or even move to another operating system a whole VG independently from other VGs.

Example 1

This covers the most common case. A device e.g. computer with a connected hard disk drive or an external USB hard disk drive or maybe something similar to these setups. However, the storage media is split into more than one partition — two in this case (note the free space in between of them — this means nothing else but a portion of the diskspace as not been partitioned[15]). What the figure also shows is the analogy with regards to the order in that the things are set up. First comes the partition. That follows the crypt device which is then used to create a file system on top of it.

Also, the figure not just only shows HDD but also USB stick, DVD and so on. That is perfectly right — you can use pretty much anything and encrypt the storage space it provides to you. I for example have encrypted my USB stick as well as a partition on my iPod. Further down that page, I will, next to others, use exactly such a setup to show you in detail how to set up and manage a system with dm-crypt[16] and LUKS[17] resp. cryptsetup to provide block-layer encryption.

The filesystem, which is installed on top of a partition can be any usual file system i.e. BtrFS, Ext2, Ext3, FAT, JFS, NTFS, OCFS2, OpenGFS, ReiserFS, Swap, XFS etc. I use XFS for the most part but will soon be a total BtrFS fan — actually I am already but then this file system is not yet (August 2008) considered finished.

Example 2

This example differs in one particular point from the former one. It uses another intermediate layer next to dm-crypt which we already know, is responsible for encryption. This additional layer is LVM (Logical Volume Manager). It allows for greater flexibility and system resilience if one decides to use the optional RAID feature set. Both, LVM and dm-crypt are, if in place, virtual block devices within the device-mapper framework of the operating system Linux kernel which are capable of mapping one block device into another — thus creating a bidirectional I/O (Input/Output) chain starting at the plain hardware disk providing storage space, upwards, through virtual block devices into userspace utilization and finally inside RAM (Random Access Memory) and again downwards onto the HDD again and so forth — it is, this is a permanent circulation controlled and managed by the kernel.

This example could be realised as follows: One might use three HDDs installed in his computer (PVs) and create LVs. Thereby, during configuration he decides to use two of the PVs to provide encrypted LVs (LV 1 and LV 2) whereas one of two is additionally configured to provide for RAID 1 (LV 1) (note the mapping from one LE into two PE on different PVs). The third PV (the third HDD in this case) is used to form a non encrypted, non RAID third LV (LV 3) with 1:1 mapping between LEs and PEs as is the case for LV 2.

All three LVs together are part of the same VG and therefore can be managed as a single unit. One could for example move the whole VG to another hardware etc. This however is a simple example. In case you use EVMS (Enterprise Volume Management System) the setup could be much more complex and feature rich.

The filesystem, which is installed on top of LVs devices, can be any usual file system i.e. Ext2, Ext3, FAT, JFS, NTFS, OCFS2, OpenGFS, ReiserFS, BtrFS, Swap, XFS etc.

Example 3

This one is just to show that things are not necessarily bound to happen on the local machine. Although figure 3 pictures a NAS (Network-attached storage), this can be any kind of storage where data stored on a remote device is accessed by another device and therefore has to travel through a computer network. On the remote site there can be any configuration — either the one depicted in figure 1 or figure 2 which might be an explicit computer configured as a NFS (Network File System) server or a dedicated network appliance[18] or any other device running a Linux kernel and thus being able to use dm-crypt.

Mostly, the traffic between remote and local machine does not need to be encrypted since this network should be considered as one to be trusted and not offering access for untrusted people or devices — it is a dedicated network for internal information exchange only, without offering access to the outside world e.g. the Internet (note, that the Internet is not to be confused with the WWW (Wold Wide Web)).

Example 4

This is when the LVM layer sits on top of the dm-crypt/encryption layer as opposed to what Example 2 shows i.e. LVM layer below encryption layer. I have a few sketches onto another page where I show how it looks like.

Theory

This section is about theory — what has to be considered before writing a single line of code for LUKS (crypsetup) or the dm-crypt device-mapper inside the Linux kernel. In essence, it is about mathematics (especially those papers written by Clemens Fruhwirth, the mastermind behind LUKS).

Author	Name/Link
Clemens Fruhwirth	New Methods in Hard Disk Encryption
Mike Peters	Encrypting partitions using dm-crypt and the 2.6 series kernel
Michael Petullo	Disk encryption in Fedora: Past, present and future
Markus Reichelt	Why Mainline Cryptoloop Should Not Be Used
Clemens Fruhwirth	TKS1 - An anti-forensic, two level, and iterated key setup scheme

For this part, I decided to just provide the reader with links to some resources since they already describe the issue plus my guess is, that most folks are not going to dive into the deep and wonderful mathematics ocean... those who will, will also prefer to choose their own travel routes thus it would be nonsense trying to provide common itineraries to individual needs and desires.

Theory and Practice

This section is probably better suited for most folks since it is pretty lightweight but should also satisfy the experienced user.

DM-Tables

Is the short for *Device Mapper Tables*. All device-mappers get there configuration from simple plaintext files. In case of the dm-crypt device-mapper, the program `dmsetup` passes on the information in its DM-Table to the kernel using the `ioctl()` system call. Well, that is all nice but did not really worked out to be user-friendly nor the best from a technical point of view

- Putting the key to decrypt the disk into a DM-Table is probably as smart as hanging your door keys on the door handle.
- The encrypted information and the information to handle the encrypted information had been separated i.e. some disk partition somewhere in the system and the DM-Table files to handle it somewhere else (they had to be plaintext and therefore storing them on the encrypted device is not possible).

This had been the case before LUKS. With the introduction of LUKS, this two disadvantages where eliminated.

Password/Key Files and Keys

Clarification of terms:

Password/Passphrase

A string of words and characters that you type in to authenticate yourself. Passphrases differ from passwords only in length. Passwords are usually short - six to ten characters. Passphrases are usually much longer - up to 100 characters or more. Their greater length makes passphrases more secure. *(The Free Computing Dictionary)* Also note, that passwords with dm-crypt are case sensitive.

Key File

A file on a computer which contains encryption or license keys. *(Wikipedia)*

Hash Function

An algorithm that turns a variable-sized amount of text into a fixed-sized output (hash value). Hash functions are used in creating digital signatures, hash tables and short condensations of text for analysis purposes (see hash buster). Hash functions are also known as *cryptographic hash functions*. *(The Free Computing Dictionary)* More on that one after the jump...

Hash resp. Hash Value

The fixed-length result of a one-way hash function. *(The Free Computing Dictionary)*

Key

A numeric code that is used to de/encrypt data for security purposes. *(The Free Computing Dictionary)*

Salt

A random number that is added to the encryption key or to a password to protect them from disclosure. *(The Free Computing Dictionary)* More after the jump...

Before LUKS and also now with LUKS, dm-crypt (resp. the Crypt Engine - see below) needs to be provided with a key of fixed length to do en/decryption to/from the encrypted virtual block device. The trick is to use a password in order to compute the key. It is so, that the password can have an arbitrary length but not the key. The key, computed from the password always has the same length (a one-way hash function is used to derive the key from the password). Actually, RIPEMD-160 is used to create the one-way hash.

A screendump to get a notion of what we are talking about:

```
sa@pc1:/tmp$ echo "fox" | md5sum
475b45b6b3531765210a51ec138af816  -
sa@pc1:/tmp$ echo "fox " | md5sum
cf1d45ef2ce7d9223ccc6aff89ff7f85  -
sa@pc1:/tmp$ echo "The fox wants to eat chicken all day long" | md5sum
16901fc60316999202f01b5c8498420f  -
sa@pc1:/tmp$ echo "fh&83ka83hz:9('#<)836hT%" | md5sum
6760de41fa137494d8ddd6da9cb14d93  -
sa@pc1:/tmp$
```

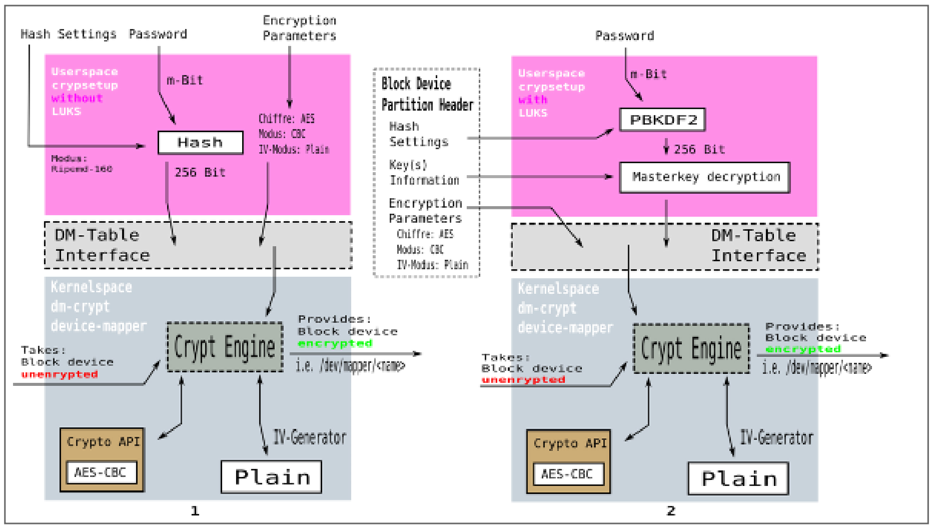
Note that the hash[19] (e.g. `475b45b6b3531765210a51ec138af816`) is always of a fixed length whereas the password (e.g. `fox)` can be of any length. Second thing to notice is how radically a little change in the password (from `"fox"` to `"fox "` (appending a blank at the end)) changes the hash. For real world usage, get used to not use passwords as simple as `fox` but rather use passwords like `fh&83ka83hz:-9('#<)836hT%`.

As of now (July 2007) `crpytsetup` on DebianGNU/Linux contains the code that brings the LUKS standard into play. In fact I do not remember for how long LUKS is already integrated into `cryptsetup` (must be a year or more) but who cares — it is there now and ready for us to use.

```
,----[ aptitude show cryptsetup | sed -n '/^Description:/,/^Tags/p' - ]
| Description: configures encrypted block devices
|  Since kernel 2.6.4, encrypted file system support is provided by the device
|  mapper target dm-crypt. This utility provides a command-line interface for
|  configuring this facility. It has integrated support for LUKS.
|
|  cryptsetup is backwards compatible with the block-layer format of cryptoloop, but
|  also supports more secure formats. This package also includes support for
|  automatically configuring encrypted devices at boot time via the config file
|  /etc/crypttab. When combined with newer versions of the debian initrd-tools and
|  standard debian kernels, cryptoroot is also supported.
|
|  For information on how to convert your system to use encrypted filesystems
|  please read /usr/share/doc/cryptsetup/CryptoRoot.HowTo.
|
| Tags: admin::boot, admin::filesystem, implemented-in::c, interface::commandline,
`----
```

The advantage, with LUKS being part of `cryptsetup` in regards to keys and passwords, is more comfort for the user and easier handling for a multi-user setup. The figure below tells about the difference:

Block-layer encryption without (1) and with (2) LUKS:



In both cases an unencrypted block device is taken, subsequently the dm-crypt device-mapper does its magic and what we get is an encrypted virtual block device e.g. `/dev/mapper/confidential` where `confidential` is the name of the virtual block device.

The difference in handling block-layer encryption with or without LUKS makes a lot of a difference. There are two dominant parts to think of:

- The encrypted information itself namely the encrypted block device e.g. `/dev/mapper/confidential`
- The information needed to handle the encrypted information i.e. `/dev/mapper/confidential` which is
    - Encryption Information e.g. `aes-cbc-essiv:sha256`
    - Password and/or keyfile
    - Hash settings

Both, the encrypted information and the information to handle the encrypted information where separated before LUKS and are in the same place now with LUKS (LUKS stores all that information within the partition header of a virtual block device).

Before LUKS (Figure 1 above)

A person had to provide all the encryption parameters over and over again each time he was mounting the virtual block device e.g. `/dev/mapper/counterintelligence`[20]. Also, there was only one password for a particular virtual block device and that in turn no possibility to provide different passwords to different people for the same virtual block device. In such case, if the only password got lost, all the data was lost as well. Changing passwords without re-encrypting all the encrypted data was not possible. This is important since the more-paranoid change passwords whenever they feel a bit followed or the corporate policy requires to change passwords on a regular and without LUKS that meant to re-encrypt TiB of data and therefore shifting around all that data — something that mostly proves impossible in practice plus it is very time consuming and simply nonsense from an economical point of view. Also, now that there is LUKS it can be seen as a disadvantage that a users password is directly used to derive the key which is then in turn used to de/encrypt the data that is read/written from/to the encrypted virtual block device. With LUKS another intermediate layer was introduced to uncouple this strict relation between a users password and the key used for encrypting the data.

With LUKS (Figure 2 above)

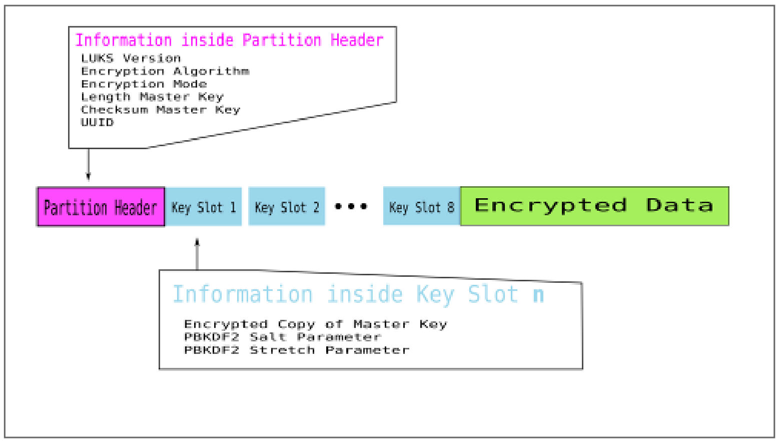
All but the password and/or keyfile is taken from the partition header of the encrypted partition. The only cumbersome thing that is left to the user is to enter a password and/or provide a keyfile.
Again, a users password is used to derive a key from it. This time PBKDF (Password Based Key Derivation Function) is used — same principle as above, which is, an arbitrary length password is used to derive a fixed length hash. This time just another procedure is used (PBKDF instead of RIPEMD-160). The key which is finally used to de/encrypt data from/to the virtual block device is not the one that is directly derived from a users password as with Figure 1 but a so called *Master Key*. What is done is that the user password is used to protect the Master Key by simply encrypting it.
The gain that comes this way is:

- A users password can be changed without the need to re-encrypt all the data.
- Since the Master Key is now encrypted it can be stored with the encrypted data. The Master Key is stored in the partition header of the virtual encrypted block device.

If somebody wants to change his user password, the current user password is used to decrypt the Master Key (Master Key is then plaintext) then the user provides his new password and the plaintext Master Key is then encrypted with the new password (becomes ciphertext again). This can be done since with either password, the plaintext Master Key stays the same and therefore there is no need the re-encrypt the whole encrypted data stored onto the encrypted virtual block device.

So with LUKS, there is a Master Key and it is stored to the partition header of the encrypted virtual block device. That is fine but there is more magic. In fact there can be more ciphertext representations of the Master Key within the partition header. Each one can then be used to de/encrypt data with the encrypted virtual block device. Each of the ciphertext representations of the Master Key can be encrypted with another password thus providing the ability to have a multi-user setup. As of now (July 2007) there are eight so called key slots.

Metainformation now part of the partition header respectively Key Slots:



The salt and stretch parameters the figure above shows are in place since humans tend to use passwords like

- Names, birth dates, etc. of their family members and the like.
- Something related to their hobby.
- Case related passwords i.e. if a person named alan had to log on to his bank account he would use `bankaccountofalan` as his password.

Quite frankly speaking, if you do that (and folks do that more than you would guess) the best on-diks encryption software and procedure would not help that in the end block-layer encryption would be frail because of the weak password.

Imagine a high security area, surrounded by a compound of impenetrable barriers and one corridor to move in and out. What are all the barriers worth when someone finds the secret key to pass the checkpoint written on the asphalt like that `Password to pass is "Hammerhead shark"` (clearly identified as key plus easy to find)? The answer is it would be worth nothing.

Salt, Stretching

In cryptography, a salt consists of random[21] bits used as one of the inputs to a key derivation function. The use of salt prevents the attackers from precomputing a dictionary of derived keys.

Salt and stretching is also to avoid the weakness of folks not using passwords like `kf&48a%la8]Q8klf43*a36jFLL-<V,RF` but things like `susanna`. So, what happens automatically is that nonetheless how *strong* a user password is, LUKS takes care and makes it really strong — using salt and stretching when deriving the key from the provided password. However, I am used to use passwords like `kf&48a%la8]Q8klf43*a36jFLL-<V,RF` at anytime, anywhere in any situation and so should you since not any software is that forward-thinking when it comes to provide a high level of security. Again, I cannot stress the fact again to use strong passphrase — on your next bus ride, collect arbitrary numbers and characters and mix them with some of `!"$%&/(/}[{)?;_>^#+*'` ... go wild... switch yourself into wild-kitten-@dance-floor-mode at that point...

Salt

The salting has to ensure that the derived key is not somehow guessable from the password or vice versa. Experts say, the derivation has to provide a high degree of entropy in order to be secure enough.

If the key for example has a length of 128 Bit (thus 2^{128} possible variations — two possible conditions (zero and one) and a sequence of 128 Bits) and the password is a lot smaller e.g. 32 Bit, we end up with (128 - 32) 96 Bit of yet *unused* space within the key. This 96 Bit could be filled with either zeros or ones (binary representation). In case we simply use `password_bits+zeros` or `password_bits+ones` we got a longer key but the key is as guessable as is the password. Since I already mentioned, folks tend to use passwords like `cat` or `butterfly` an attacker could do the following in order to attack/guess the key.

> At a side note, the reader might want to read about the differences of variation, combination and permutation in order to understand what we are dealing with.

An Example

Words like `cat`, `butterfly` etc. can be found in dictionaries. So the attacker respectively his computer would just have to try all words in a dictionary with your encrypted hard disk drive. You may think that is no problem since a dictionary has so many words written in it. Well... we are talking about computers here not human beings carrying out this task.

> *The largest dictionary in the world is "het Woordenboek der Nederlansche Taal (WNT)" (the Dictionary of the Dutch language). It took 134 years to create the dictionary (1864 - 1998). It consists of approximately 400,000 words on 45805 pages in 92000 columns. (Wikipedia)*

So, 400,000 words... that is a lot from a humans point of view but we need to take another point of view:

```
sa@pc1:~$ python
Python 2.4.4 (#2, Apr 26 2007, 00:02:45)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2**19
524288
>>>
sa@pc1:~$
```

So with 19 Bits (2^{19}), what we got is around half a million different keys (words in the dictionary). All we have to do is to write a program that uses all those million words in a dictionary in order to try them as the key for the encrypted hard disk. We are lucky since we do not have to write such a program — there are already many of them on the net, ready for anybody to download and use. The problem is Mr. or Mrs. *anybody* could also be your worst enemy... As a matter of fact, positively attacking a 20 Bit or so key is possible for anybody with of-the-shelf hardware and even more easily if one has custom hardware available.

PBKDF (Password Based Key Derivation Function) takes care about this in simple but ingenius manner — the computation in order to derive the key from the password has intentionally been made very CPU-intensive. That is absolutely no problem since you, who knows the correct password/keyfile can provide it and subsequently the key gets computed and that is it. Even if it takes one or two seconds (that is very long) it does not really matter. It matters if you had to try it 524288 or 2^19 times. If you have hardware that manages to compute the key in one second, trying to guess the key depends on the key length — thus, it could take up to

```
sa@pc1:~$ python
Python 2.4.4 (#2, Apr 26 2007, 00:02:45)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 524288 / 60
8738
>>> 524288 / (60**2)
145
>>> 524288 / ((60**2) * 24)
6
>>> 2**19 / ((60**2) * 24)
6
>>> 2**20 / ((60**2) * 24)
12
>>> 2**21 / ((60**2) * 24)
24
>>> 2**128 / ((60**2) * 24)
3938453320844195178974243141571391L
>>> 2**128 / (((60**2) * 24)* 365)
10790283070806014188970529154990L
>>>
sa@pc1:~$
```

As can be seen, half a million attempts would take around six days. It is about twice the time if we just take one more Bit (twenty Bits). With 21 Bits, it would already take 24 days. The last two lines show how long it might take to guess a key with 128 Bits — well, that is more than the age of the universe so I guess a curious person may need a lot of patience. Of course, that example is pretty rough i.e. we did not bring into play mathematics e.g. statistics, probability etc. but for now that is fine and enough to get a notion about how much the key length influences the overall security (secrecy in this case) of encrypted data stored onto a storage device like a hard disk drive for example.

Another advantage that comes with salt is the fact, that if for example two passwords were provided to the KDF (Key Derivation Function) one after another do not result in the same hash (key). That, next to other things, is part of a procedure providing a high degree of entropy.

Stretching

Again, it is all about the beauty of mathematics i.e. Combinatorics in this case. After the KDF provided us with a high degree of entropy by *salting* the derivation from the key to the password the user provided, the whole key derivation process is still vulnerable for attacks given enough resources (money, hardware, time, people, brain power, etc.) to the attacker. At this point LUKS introduces stretching. After say 64 of 128 Bit have been mangled by applying salt, the rest of the Bits (128 - 64) which is 64 in this case is not simply padded with zeros or ones but with another random (pseudo-random) value in order to avoid the following.

Example:

An attacker could pre-compute all keys (words in a dictionary), store them and later use the stored hash values for guessing the key. In this case he might only compute the hashes (keys) once and later on use them over and over again which reduces the effort from say 6 days to a couple of seconds or so in case of 19 Bits — as you see, this a huge factor; this factor makes it possible to guess the key which in turn renders block-layer encryption useless and probably ensures that ones nightmares become a reality by revealing what should be a secret.

Above, I mentioned that programms to pre-compute all keys in a dictionary exist — just search for them and you will find many of them... but why? You do not have to wait a long time to compute all the keys for different dictionaries since that has already been done — again, search the net and download the result of the computations a brave attacker has already carried out. This pre-computed tables containing all hash keys can be used by any kid in order to attack you, your business etc. With LUKS this is no problem since LUKS takes care about such vulnerability by using salt and padding the key with a pseudorandom value.

With stretching dictionary attacks are impossible since one would have to compute

```
sa@pc1:~$ python
Python 2.4.4 (#2, Apr 26 2007, 00:02:45)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2**19
524288
>>> 2**(128-19)
649037107316853453566312041152512L
>>> (2**(128-19))*(2**19)
340282366920938463463374607431768211456L
>>> 2**128
340282366920938463463374607431768211456L
>>>
sa@pc1:~$
```

not just all possible keys (words in the dictionary) which we already know is (2^{19}) but ANY combination of any key with any number that can be formed of the (128-19) 109 Bits which we already know is ($2^{19} * 2^{109} = 2^{128}$) respectively this absurd long decimal number above starting with `34`. It is simply impossible to compute all combinations and try guessing the key thus also pre-computed key/hash tables turn out to be no danger for LUKS. I repeat... not possible for no-one that includes not just individuals but also any company, government, secret service etc. since there are more possible combinations than atoms in the universe.

Destroy Data, IT Forensics

This section provides a quick look into IT forensics and how it works plus even more importantly, how private persons, companies, governments, military etc. can protect themselves against state of the art techniques uses in IT forensics.

Visitors, Reselling, Dispose

It is common with IT (Information Technology) forensics (e.g. http://www.forensicts.co.uk/ - contractors to Scotland Yard and the like) that HDDs (Hard Disk Drives) are the most important source of information next to network traffic. Whatever you consider a secret, if you use block-layer encryption with dm-crypt then you can be sure that even if your hardware is taken away from you by force, you are save since those folks cannot decrypt the data — thus what is a secret stays a secret as long the password is a secret as well.

Another, more practical use case, is the one where old storage media is replaced by newer one. Without block-layer encryption this storage media cannot simply be put on resell or thrown away — anybody could *read* the data... Instead, one would have to use tools like `shred`, `secure-delete`, `wipe` etc. or some home-made stuff like using `dd` for overwriting the whole storage media several times with a random sequence of zeros and ones. *This (the exchange of old with new storage media) is a big issue for all kinds of companies, governmental institutions, the military, the health system (confidential records about a persons medical history), etc.*

Time matters

Hard disks drives have a very long remaining memory effect. Even if data appears to be gone, even if the disk has been reinitialised with zeros, even if you invoked the security-erase command of your IDE hard disk, data can be easily recovered. Neither ATA (Advanced Technology Attachment) nor SCSI (Small Computer System Interface) instructions sets provide the opportunity to overwrite old physical sectors which had been in use so far but it then happened that the virtual sector got remapped to yet another physical sector.

Currently (July 2007) believes are that every single Bit has to be changed (overwritten) at least 40 times in order to withstand state of the art IT forensic efforts to *read* data from a storage media. Running `shred` on a ~233GiB (~250GB) takes two days or so. By default `shred` overwrites each Bit 25 times so I assume overwriting 233GiB forty times would take three days or more.

This (the time it takes to overwrite a HDD several times) plus the fact that there are already TiB HDDs available should make it clear to anybody that attempts to *destroy* information (data as the representation of information) this way (by deleting/overwriting) is nothing but a blind alley.

It is a no brainer that there is neither the time to overwrite data when the doorbell already rings and some curious folks ask you weird stuff nor is there the time to overwrite tens or hundreds of HDDs which are scheduled for disposal or resell.

The only way out of this misery is to use block-layer encryption since with block-layer encryption your data is secure at any times starting with the setup of dm-crypt. If the curious then knock on your door, all you have to do is to pull the plug or if you want the resell your HDD, just do it without concerns — in both cases only you know the password...

Destroying the Key

Assuming we use block-layer encryption with LUKS and dm-crypt, the only thing left to us is to *destroy* the key we would normally use to decrypt data stored within a virtual block device. This however is a bit tricky since some storage media e.g. HDDs are conceptually designed to avoid the loss of data (which of course is perfectly right). Extra precaution must be taken for this data, since data is not guaranteed to be ever erased.

> Note, that destroying (deleting) the key is a second security net since even though the attacker(s) might somehow manage to know the password, if the key has been destroyed, he cannot decrypt the data — only the password AND the key would allow to decrypt the data.

However, the problem is that the key to unlock the encrypted virtual device is stored to key slots (see above) which fit into a single sector on the disk. The firmware carries out so called sector remapping[23]. Unintended remapping of sector data is especially bad for key material created via a key hierarchy, as the encrypted key material is typically very short and can be accommodated by a single sector in a reserved remapping zone.

So, if for example, the HDDs firmware remaps a sector and we try to overwrite the sector afterwards with e.g. `dd`, still, the old sector with the key remains onto the HDD. This sector can then be used to decrypt the data stored onto the virtual block device since all commands get redirected to the new physical sector (which is the same virtual sector as it had been before; thus it is called *mapping* virtual to physical sectors).

In this case, an attacker (and be sure these guys have the resources, knowledge and will) can equip the HDD with a modified firmware in order to reread old (yet unused) physical sectors again.

LUKS (Linux Unified Key Setup) takes care of the afore mentioned issue as well. How this is accomplished follows subsequently.

AF Splitting and AF Merging

The intention of *AF Splitting* and *AF Merging* is to not allow IT forensics to recover the key respectively to ensure the key can be securely destroyed if we want to. Such methods are commonly known as AF (Anti-Forensic).

As we already know, in order to make sure we can securely (no chances to recover) destroy the key on a persistent storage media, using methods like `shred` (see above) and the like are simply unemployable. To be more precisely, the AF (Anti-Forensic) method is called AF (Anti-Forensic) Information Splitting/Merging.

The smart thing with the AF methods being part of LUKS (Linux Unified Key Setup) is as follows.

The Splitting

What *AF Splitting* does is quite simple but genius at once. It takes the master-key data and balloons it to about 4000 times of its original size. Roughly speaking, because of the new size, it is necessary to distribute the master-key (here, denoted with the letter x) across the whole HDD (Hard Disk Drive). This leads to the fact that the key is then stored in about 4000 sectors[24].

The way those 4000 chunks ($x = a0 + a1 + a2 + \ldots + a3999$) are computed is so, that a function computes a from $a - 1$. This function ensures for each a

- That the result (a) is random thus leading to the fact that
- Any a it is unique - thus no redundancy can be found across all as

The Merging

At some point, the complete master-key (what it has been before ballooning) is of course needed to unlock/decrypt data stored within the encrypted virtual block device. Any data can be securely destroyed if its not stored in persistent but volatile storage media. Since HDDs are persistent storage media and RAM (Random Access Memory) is not (its volatile) the following is done.

AF Merging is the process of reading all the sectors, each containing a chunk of the 4000 original master-key chunks from HDD into memory and then merging it — all merging takes place inside a computers RAM which guarantees (as with in contrast to HDDs) that it can be deleted securely.

If only one of the 4000 chunks contains incorrect data or is absent, the *AF Merging* and thus AF (Anti-Forensic) fails entirely.

Bottom Line

Because it can be statistically shown, that overwriting one out of 4000 sectors on a HDD (Hard Disk Drive) can be done with almost 100% probability, the *AF Splitting/Merging* method ensures to be secure against AF (Anti-Forensic) methods and those who use it to reveal your secrets, those of or your business or your country or...

In conjunction with key hierarchies and PBKDF (Password Based Key Derivation Function), LUKS (Linux Unified Key Setup) provides a sophisticated key/password management system which allows to add, delete and change key/keyfiles the secure way (no way for AF (Anti-Forensic) to recover a key).

Ciphers, Chaining Mode, IV, Watermark

With the CLI (Command Line Interface) tool `cryptsetup`, there is the possibility to not just name a particular cipher for the virtual block device but to also choose the chaining mode as well as the IV (Initialization Vector) e.g. `-c aes-cbc-essiv:sha256`. As we can see, those three are separated by hyphens with `-c` being a crpytsetup switch.

Ciphers used by Dm-Crypt

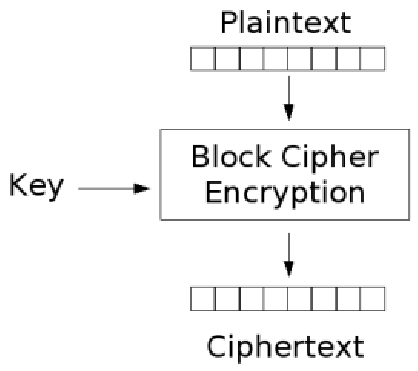
To really understand dm-crypt and LUKS, it is necessary to take a look at the principles that play a major role when it comes to block-layer encryption. We need to know about the basics of the techniques in place in order to understand what is going on, how it is works and why it is done in a particular way. What exactly is a cipher anyway?

Cipher

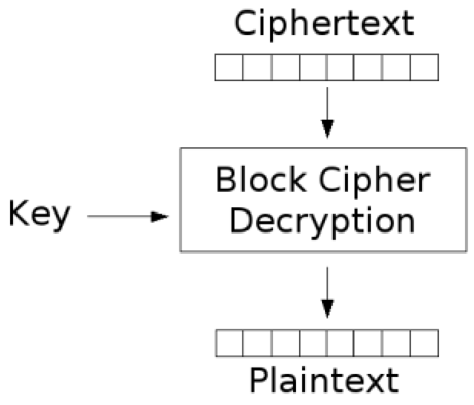
> A cipher is an algorithm for performing encryption. The reversible transformation of data from the original (the plaintext) to a difficult-to-interpret format (the ciphertext) as a mechanism for protecting its confidentiality, integrity and sometimes its authenticity. Encryption uses an encryption algorithm and one or more encryption keys. See encryption algorithm and cryptography. *(The Free Computing Dictionary)*

Block Cipher vs Stream Cipher

In cryptography, a block cipher is a symmetric key cipher which operates on fixed-length groups of bits, termed blocks, with an unvarying transformation. When encrypting, a block cipher might take a (for example) 128-bit block of plaintext as input, and output a corresponding 128-bit block of ciphertext. The exact transformation is controlled using a second input — the secret key. Decryption is similar: the decryption algorithm takes, in this example, a 128-bit block of ciphertext together with the secret key, and yields the original 128-bit block of plaintext.



Encryption



Decryption

This is in contrast to how a stream cipher works:

In cryptography, a stream cipher is a symmetric cipher where plaintext bits are combined with a pseudorandom cipher bit stream (keystream), typically by an XOR (exclusive-or) operation. In a stream cipher the plaintext digits are encrypted one at a time, and in which the transformation of successive digits varies during the encryption. An alternative name is a state cipher, as the encryption of each digit is dependent on the current state. In practice, the digits are typically single bits or bytes.

Stream ciphers represent a different approach to symmetric encryption from block ciphers. Block ciphers operate on large blocks of digits with a fixed, unvarying transformation.

Because of its nature (dm-crypt works on data blocks), dm-crypt makes use block ciphers and not of stream ciphers.

Block Cipher Modes of Operation

Now, that we know that dm-crypt makes use of block ciphers only, there is still more to know. Block ciphers know several modes of operation. It is not only the cipher in use but also the operation mode the cipher works in that decides what ciphertext is written to the storage media and thus it also has big impact on how secure block-layer encryption is.

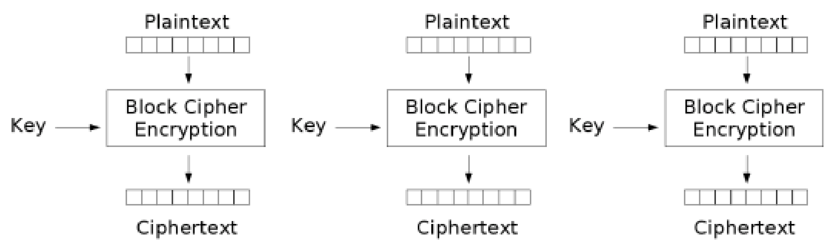
Dm-crypt currently (July 2007) allows for two different operation modes:

- ECB (Electronic Code Book)
- CBC (Cipher Block Chaining)

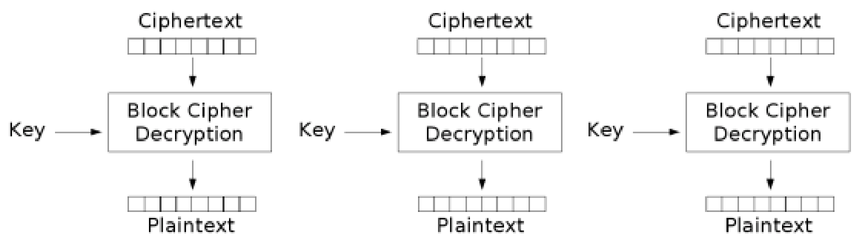
CBC is preferable (see below why this is true) over ECB but still, both suffer from some security issues which would vanish if LRW-AES (LRW: Liskov, Rivest, Wagner; AES: Advanced Encryption Standard) would be implemented. This however is not done yet because of some issues[26] with the Linux Kernel itself.

Electronic Code Book

The simplest of the encryption modes is the electronic codebook (ECB) mode. The message is divided into blocks and each block is encrypted separately. *The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks* — thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.



Electronic Codebook (ECB) mode encryption



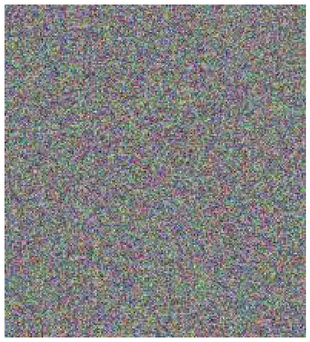
Electronic Codebook (ECB) mode decryption

As mentioned, identical plaintext block lead to identical ciphertext block — this is less secure than what CBC (Cipher Block Chaining) does. Below is a striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext. A pixel-map version of the image on the left was encrypted with ECB mode to create the center image and with CBC to create the right image:

Tux Original

ECB Encryption

ECB Encryption



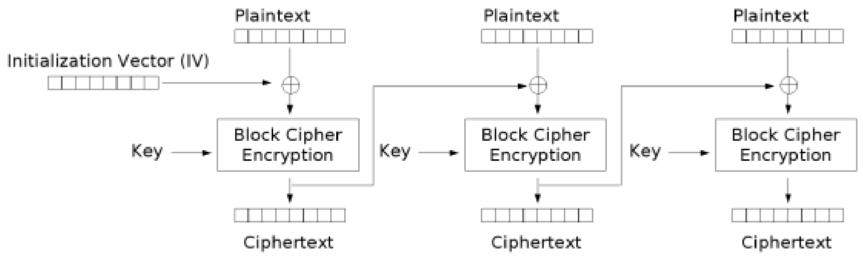
CBC Encryption

The image on the right is how the image might look like if it had been encrypted with CBC or any of the other more secure modes — indistinguishable from random noise. The center image shows, that same input (plaintext) results in same output (ciphertext) thus ECB is less secure than CBC.

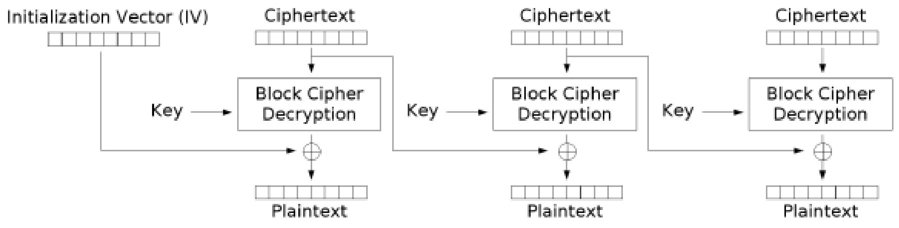
As you might understand now, how secure block-layer encryption is depends not just only on the cipher used but also on the operation mode — the best cipher may produce insecure results if used with a weak mode of operation.

Cipher Block Chaining

In the CBC mode, each block of plaintext is XORed (exclusive-or) with the previous ciphertext block before being encrypted. This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size[27].

Note that a one-bit change in a plaintext affects all following ciphertext blocks, and a plaintext can be recovered from just two adjacent blocks of ciphertext. As a consequence, decryption cannot be parallelized, and a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext.

The parallelization issue is one issue that would vanish with the implementation of LRW-AES — this would allow to speed up block-layer encryption on multiprocessor hardware since LRW-AES can do parallel computation of plaintext to ciphertext blocks which is impossible with CBC since block **n** can only be computed if **n-1** has already be done.

Available Ciphers

One can choose different ciphers when it comes to switch data between its plaintext respectively ciphertext representation in order to unhide/hide information.

We can take a look at the available ciphers

```
sa@pc1:~$ cat /proc/crypto
name         : sha256
driver       : sha256-generic
module       : sha256
priority     : 0
refcnt       : 1
type         : digest
blocksize    : 64
digestsize   : 32

name         : cbc(aes)
driver       : cbc(aes-generic)
module       : cbc
```

```
priority     : 100
refcnt       : 3
type         : blkcipher
blocksize    : 16
min keysize  : 16
max keysize  : 32
ivsize       : 16

name         : aes
driver       : aes-generic
module       : aes
priority     : 100
refcnt       : 5
type         : cipher
blocksize    : 16
min keysize  : 16
max keysize  : 32

name         : md5
driver       : md5-generic
module       : kernel
priority     : 0
refcnt       : 1
type         : digest
blocksize    : 64
digestsize   : 16

sa@pc1:~$
```

Those (the above) are the ciphers which are compiled into the Linux kernel and can be used. Since I use a prepackaged kernel binary,

```
sa@pc1:~$ uname -a
Linux pc1 2.6.21-2-vserver-686#1 SMP Mon Jun 25 23:45:40 UTC 2007 i686 GNU/Linux
sa@pc1:~$
```

the ciphers I may use are fix but that is ok with me since `sha256` does a pretty good job.

One is either satisfied with the possible choices or has to compile his own kernel with additional ciphers. In case he would like to add additional ciphers, he could use `make menuconfig` and then go to `Cryptographic Options --> Cryptographic API` and choose those ciphers he would like to have available and then recompile his kernel.

Initialization Vector

An IV (Initialization Vector) is a block of bits that is required to allow a stream cipher or a block cipher to be executed in any of several modes of operation to produce a unique ciphertext output independent from other outputs produced by the same encryption key, without having to go through a (usually lengthy) re-keying process.

All block cipher operational modes (except ECB (Electronic Code Book)) require an IV (Initialization Vector) — a sort of *dummy block* to kick off the process for the first real block, and also to provide some randomization for the process. There is no need for the IV to be secret, in most cases, but it is important that it is never reused with the same key.

For CBC (Cipher Block Chaining) and CFB (Cipher FeedBack), reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. If the IV is chosen at random (which is the case with dm-crypt), the cryptographer must take into consideration the probability of collisions, and if an incremental IV is used as a nonce, the algorithm's resistance to related-IV attacks must also be considered.

For OFB (Output FeedBack) and CTR (Counter), reusing an IV completely destroys security. In CBC mode, the IV must, in addition, be randomly generated at encryption time. *(Wikipedia)*

More information after the jump...

Redundancy, the Watermarking Attack and its Countermeasures

Redundant Data may lead to successful Watermarking Attacks

Disk encryption suites generally operate on data in 512-byte sectors which are individually encrypted and decrypted. These 512-byte sectors alone can use any block cipher mode of operation (typically CBC), but

since arbitrary sectors in the middle of the disk need to be accessible individually, they cannot depend on the contents of their preceding/succeeding sectors. Thus, with CBC, each sector alone has to use an IV (Initialization Vector).
At this point you might be a little confused since before,it has been said that with CBC (Cipher Block Chaining), to encrypt data written to e.g. HDD (Hard Disk Drive) it is necessary go trough a chain of blocks, encrypting not yet encrypted (thus called plaintext) blocks with their adjacent ciphertext blocks.
That is correct but also, as an countermeasure against watermarking attacks (more on that subsequently), an IV is applied any time a plaintext block is written (and thereby encrypted) to the storage media (the virtual block device on top of e.b. an USB (Universal Serial Bus) stick, HDD, etc.).

```
`n'        total number of encrypted sectors
`a'        sector index
`x = a-1'  number of ciphertext sectors before sector `a'

ENCRYPTION:
Each sector `a' depends on `a-1' for encryption -- using the
ciphertext of `a-1' and an IV (Initialization Vector).

DECRYPTION:
In case of decryption, to decrypt ciphertext sector `a', it is only
necessary to have ciphertext sector `a-1' around thus the operation is
in O(1). It does not require to re-encrypt a number `x' of sectors
each time a sector `a' is decrypted thus making it O(n).
```

If these IVs are predictable by an attacker, then a specially crafted file can be generated (containing redundant code) to NOP-out the IV, causing different blocks on the encrypted disk to have identical sectors, or at least the first block in a number of sectors to be identical (this is what the watermarking attack is about).

The sector patterns generated in this way can give away the existence of the file, without any need for the disk to be decrypted first[28]. The problem is analogous to that of using block ciphers in the electronic codebook (ECB) mode (remember the Tux images above), but instead of whole blocks, only the first block in different sectors are identical.

Countermeasures

Watermarking attacks are possible if, and only if, the IV (Initialization Vector) is predictable. What needs to be done, is to make it unpredictable. There are two different approaches to counter successful watermarking attack.

1. The first one is to include a unique information per HDD sector with the IV. The sector number (position) is such a unique data pattern. So, even if the attacker provides us with a file of redundant code, containing repeating patterns of the size of an sector or smaller, the resulting ciphertext block would be different for each plaintext block.
2. Another approach would be to use ciphers in modes of operation which already take into account earlier encrypted plaintext blocks (see example `ENCRYPTION` above). CBC (Cipher Block Chaining) does a pretty good job in this regard — it takes the ciphertext of $a-1$ and does an XOR operation with $a-1$ (already ciphertext at that point) and a (still plaintext at that point). After the XOR operation, the resulting block (containing a bunch of bits) is not instantly written to the storage media at sector position a but the block is then encrypted with the cipher in action and then written to the storage media at position of sector a and thus becoming a new ciphertext block. In simple words, this is a two-stage procedure — STEP ONE: the XOR; STEP TWO: encrypting the outcome of step one.

In both cases, watermarking attack are becoming impossible since the IV (Initialization Vector) cannot be predicted.

Encrypted Salt-Sector Initialization Vector

ESSIV (Encrypted Salt-Sector Initialization Vector) is a method for generating initialization vectors for block encryption to use in disk encryption. The usual methods for generating IVs are predictable sequences of numbers based on for example time stamp or sector number and permits certain attacks such as a Watermarking attack.

ESSIV on the other hand generates the IV from a combination of the sector number with the hash of the key. It is the combination with the key in form of a hash that makes the IV unpredictable.

Combination of CBC and ESSIV

Finally, if one chooses to use `aes-cbc-essiv:sha256`, it tells us he is using `cbc` (mangling the IV via $a-1$ ciphertext) and `essiv` (mangling the IV via $a-1$ ciphertext and the secret key).

As I mentioned above, there are two methods to avoid redundant data patterns

- using the sector number of a to compute the IV
- using the ciphertext of $a-1$

The fact that `aes-cbc-essiv:sha256` uses both makes it the most secure variant that is currently (July 2007) possible. It has already been mentioned that LRW-AES (LRW: Liskov, Rivest, Wagner; AES: Advanced Encryption Standard) is not yet implemented.

Performance

The questions narrows down to two questions:

Being with or without block-layer encryption

As can be seen with the figures above, with block-layer encryption there is another intermediate layer around. It is quite clear that it takes time to do the computations between the ciphertext and plaintext domain. Because of that, block-layer encryption is of course *slower* (has a lower HDD (Hard Disk Drive) I/O (Input/Output)) than a system without block-layer encryption.
I am not going to do some extended measurements across several systems once with and once without dm-crypt being used because I think this is just of academical interest. For practical reasons, I can tell you what I can do with of-the-shelf hardware with block-layer encryption in place.
I can watch a movie, listen to some mp3s and perform a system backup with `rsync` — all at the same time onto/from an external USB (Universal Serial Bus) HDD which is entirely encrypted with dm-crypt without noticing any sort of delays and the like. Finally, to show you some numbers

```
sa@pc1:~$ cat /proc/cpuinfo | egrep 'bogo|name|cache'
model name      : Intel(R) Pentium(R) 4 CPU 3.06Gz
cache size      : 512 KB
bogomips        : 6138.51
sa@pc1:~$ su
Password:
pc1:/home/sa# cryptsetup status /dev/mapper/alan
/dev/mapper//dev/mapper/alan is active:
  cipher:  aes-cbc-essiv:sha256
  keysize: 128 bits
  device:  /dev/sdd1
  offset:  1032 sectors
  size:    781416570 sectors
  mode:    read/write
pc1:/home/sa# hdparm -tT /dev/mapper/alan

/dev/mapper/alan:
 Timing cached reads:   462 MB in  2.00 seconds = 230.93 MB/sec
 Timing buffered disk reads:   38 MB in  3.05 seconds =  12.45 MB/sec
pc1:/home/sa# exit
exit
sa@pc1:~$
```

Of course, a virtual block device residing on top of an RAID (Redundancy Arrays of Independent Disks) array which is connected to the PCIe (PCI Express) bus via an RAID HBA (Host Bus Adapter) delivers much higher I/O (Input/Output) values than what I got above (`12.45 MB/sec`) with my single external USB HDD with 16MB cache memory.

However, those `~12MB/sec` I get is more then enough to do what I described above and much more important — what I consider a secret stays a secret no matter what happens! For me, not using block-layer encryption is simply not an option. Remember what I wrote above — it is all about self-determination...

Which block-layer encryption setup is the fastest?

As the screendump above shows, I am currently using `aes-cbc-essiv:sha256` which I do because to me, security has a higher priority than performance. Fact is, that my current setup (and I have chosen the slowest possible case to demonstrate it = external USB HDD) is more than fast enough for daily usage. If LRW-AES

(LRW: Liskov, Rivest, Wagner; AES: Advanced Encryption Standard) would be available (hopefully it will be soon) I will switch over using it, again making security a higher priority than performance.

Ciphers and Modes of Operation

1. It is not just the cipher itself that determines how secure block-layer encryption is but also the mode of operation the cipher operates in.
2. CBC (Cipher Block Chaining) and ECB (Electronic Code Book) can be used — CBC is more secure than ECB. LRW-AES (LRW: Liskov, Rivest, Wagner; AES: Advanced Encryption Standard) is not yet (July 2007) implemented but would be preferable over CBC for performance (possible usage of multiprocessor hardware) and stronger security.
3. ESSIV (Encrypted Salt-Sector Initialization Vector) is the most secure way to create an IV (Initialization Vector) — it prevents successful watermarking attacks.
4. Several ciphers e.g. `sha256` may be used out of the box or via recompiling the Linux Kernel.
5. The combination of `cbc-essiv` as in `-c aes-cbc-essiv:sha256` makes sense.
6. I would recommend that one should assign security a higher priority than performance when it comes to decision making since in practice it turns out that also the most *secure cipher + operation mode combination* is fast enough — in a multiprocessor environment it can even be faster than a less secure variant would be (recall what I wrote about LRW-AES and parallelism).

Setup and Manage

This section is all about getting to grips with block-layer encryption. It is split into several subsections in order to cover different approaches as discussed above with the three examples.

Prepare

This subsection is about preparations — what needs to be in place in order to start setting up block-layer encryption with some storage media.

Hardware

We need to have the storage media (e.g. HDD (Hard Disk Drive) or USB (Universal Serial Bus) stick etc.). This storage media will either be used entirely (encrypting the whole storage space) or just partially.

Since setting up block-layer encryption with dm-crypt and random storage media involves overwriting the storage space on the storage media, be aware that all data is irretrievably lost in the course of setting up the virtual block device. Because of that, if the storage media is not yet unused by you but contains data you might want to do a backup to another storage media. Later on, after setting up dm-crypt, you can move the data back onto the storage media — then being outfitted with the dm-crypt device-mapper virtual block device and thus providing transparent en/decryption for I/O (Input/Output) to/from that storage media.

You also need to connect the storage media to a computer or some sort of embedded device built into your pool-cleaning-bot or whatever in order to carry out the setup procedure for dm-crypt. Later on, to use block-layer encryption, the storage media has to be constantly connected to a computer or your bot etc. if you want to use block-layer encryption.

So, it is quite common that folks use one external USB (Universal Serial Bus) casing with more than one HDD (Hard Disk Drive) to do backups with `rsync` or `dar` for example. Say one has got three HDDs and one USB casing — he could then set up block-layer encryption on all three HDDs and cycle through the HDDs (putting the next one into the casing) every week or so and use `rsync` to do a backup. Of course, preferably and common with business setups is to set up block-layer encryption with dm-crypt on top of an RAID (Redundancy Arrays of

Independent Disks) device running a bunch of disk within decent 19" rack-mounted hardware or even better use dm-crypt with a dedicated storage appliance.

Software

Next step after ensuring that the desired hardware is in place, we need to acquire all the software needed to set up and run block-layer encryption with dm-crypt. Basically that means we need to have two things before we can actually start with the setup

- a Linux Kernel with enabled support for dm-crypt
- the userspace tool(s) to set up and manage encrypted storage media

The Kernel

If you are with DebianGNU/Linux, all you need to do is to install a kernel image via Debians outstanding package management system called `libapt` or better know as APT (Advanced Packaging Tool) since those kernel packages already come with enabled dm-crypt support. Which front-end you use is fully up to you. Since I am a person who, for some good reasons, prefers the CLI (Command Line Interface) over GUI (Graphical User Interface). I usually use some of the `apt-*` variants e.g. `apt-get` or `aptitude` in its CLI mode. For those who prefer GUIs, `aptitude` also features a GUI plus there is `gnome-apt`. Note, for the subsequent examples I am going to use command aliases so do not be confused if `acsn` for example does not ring a bell since it is just an alias for

```
,----[ cat ~/.bashrc | grep acsn ]
| alias acsn='apt-cache search --names-only'
`----
```

If you want to figure the current pre-build kernel packages with Debian you can do that

```
sa@pc1:~$ date -u
Tue Jul 31 09:36:24 UTC 2007
sa@pc1:~$ finger @kernel.org
[kernel.org]
The latest stable version of the Linux kernel is:           2.6.22.1
The latest prepatch for the stable Linux kernel tree is:    2.6.23-rc1
The latest snapshot for the stable Linux kernel tree is:    2.6.23-rc1-git9
The latest 2.4 version of the Linux kernel is:              2.4.35
The latest 2.2 version of the Linux kernel is:              2.2.26
The latest prepatch for the 2.2 Linux kernel tree is:       2.2.27-rc2
The latest -mm patch to the stable Linux kernels is:        2.6.23-rc1-mm1
sa@pc1:~$ acsn linux-image-2.6
linux-image-2.6-486 - Linux 2.6 image on x86
linux-image-2.6-686 - Linux 2.6 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6-686-bigmem - Linux 2.6 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6-amd64 - Linux 2.6 image on AMD64
linux-image-2.6-k7 - Linux 2.6 image on AMD K7
linux-image-2.6-vserver-686 - Linux 2.6 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6-vserver-k7 - Linux 2.6 image on AMD K7
linux-image-2.6.21-2-486 - Linux 2.6.21 image on x86
linux-image-2.6.21-2-686 - Linux 2.6.21 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.21-2-686-bigmem - Linux 2.6.21 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.21-2-amd64 - Linux 2.6.21 image on AMD64
linux-image-2.6.21-2-k7 - Linux 2.6.21 image on AMD K7
linux-image-2.6.21-2-vserver-686 - Linux 2.6.21 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.21-2-vserver-k7 - Linux 2.6.21 image on AMD K7
linux-image-2.6.22-1-486 - Linux 2.6.22 image on x86
linux-image-2.6.22-1-686 - Linux 2.6.22 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.22-1-686-bigmem - Linux 2.6.22 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.22-1-amd64 - Linux 2.6.22 image on AMD64
linux-image-2.6.22-1-k7 - Linux 2.6.22 image on AMD K7
linux-image-2.6.22-1-vserver-686 - Linux 2.6.22 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.22-1-vserver-k7 - Linux 2.6.22 image on AMD K7
linux-image-2.6.22-rc7-486 - Linux 2.6.22-rc7 image on x86
linux-image-2.6.22-rc7-686 - Linux 2.6.22-rc7 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.22-rc7-686-bigmem - Linux 2.6.22-rc7 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.22-rc7-amd64 - Linux 2.6.22-rc7 image on AMD64
linux-image-2.6.22-rc7-k7 - Linux 2.6.22-rc7 image on AMD K7
sa@pc1:~$
```

The first few lines are merely because I just wanted to show that Debians pre-build kernel packages are not outdated but pretty much up-to-date — you can see, the current stable Linux kernel is at version `2.6.22.1` and Debian provides `linux-image-2.6.22-rc7-amd64` for example. Folks moaning about thus fact are mostly greenhorns anyway and simply do not know better. If you want to install pre-build `-rc` kernel packages you might add

```
,----[ grep -A1 '^# kernel' /etc/apt/sources.list ]
| # kernel snapshots - sid
| deb http://kernel-archive.buildserver.net/debian-kernel sid main
| --
| # kernel snapshots - trunk (= most up-to-date and risky)
| deb http://kernel-archive.buildserver.net/debian-kernel trunk main
`----
```

to your `sources.list`. Note, you could have also used aptitude — whatever you use, stick with it. Switching back and forth between `apt-get` and `aptitude` might cause problems since both leave different traces within the package database, problematic for the other one to work well with.

```
sa@pc1:~$ aptitude search linux-image-2.6.*vserver
p   linux-image-2.6-vserver-686                   - Linux 2.6 image on PPro/Celeron/PII/PIII/P4
p   linux-image-2.6-vserver-k7                    - Linux 2.6 image on AMD K7
c   linux-image-2.6.17-2-vserver-686              - Linux 2.6.17 image on PPro/Celeron/PII/PIII/P4
c   linux-image-2.6.18-4-vserver-686              - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
c   linux-image-2.6.20-1-vserver-686              - Linux 2.6.20 image on PPro/Celeron/PII/PIII/P4
i   linux-image-2.6.21-2-vserver-686              - Linux 2.6.21 image on PPro/Celeron/PII/PIII/P4
p   linux-image-2.6.21-2-vserver-k7               - Linux 2.6.21 image on AMD K7
C   linux-image-2.6.22-1-vserver-686              - Linux 2.6.22 image on PPro/Celeron/PII/PIII/P4
p   linux-image-2.6.22-1-vserver-k7               - Linux 2.6.22 image on AMD K7
sa@pc1:~$
```

However, no matter what pre-build kernel package you install, all come with dm-crypt and LUKS (Linux Unified Key Setup) support enabled

```
,----[ grep DM_CRYPT /boot/config-2.6.22-1-vserver-686 ]
| CONFIG_DM_CRYPT=m
`----
```

As you can see, I am currently on

```
,----[ uname -a ]
| Linux pc1 2.6.22-1-vserver-686 #1 SMP Mon Jul 30 03:18:43 UTC 2007 i686 GNU/Linux
`----
```

Finally, if you want to or have to for some special reason, you can always get the vanilla sources from `kernel.org` and recompile the latest and greatest e.g. `2.6.23-rc1-mm1` as it is as of now (Tue Jul 31 09:59:11 UTC 2007). The downside of the latest and greatest is of course that you are pretty much running untested code which in turn means it could have devastating effects — could screw up the box running your business or even worse, it might kill your cat. Best choice for production environments would be to choose the stable Debian branch with well-tested userland and kernels plus being provided with instant security fixes by the Debian community — nothing of what you get with the *latest and greatest*...

See,
with free software,
it is YOU you who has the power of choice...

Now that we know what pre-build kernel binaries are available, we need to install what we considered suitable for our environment. I am going with current DebianGNU/Linux unstable branch (aka sid (still in development)) kernels on my workstation and mobile gadgets. If you would like to install a pre-build kernel image, you would have to issue e.g. `aptitude install linux-image-2.6.22-1-vserver-686` or `apt-get install linux-image-2.6.22-1-vserver-686` to use the CLI or go with `gnome-apt` if you have a liking for GUIs.

> We are using pre-build Debian kernel binary packages with our current efforts to set up block-layer encryption — I will not cover the procedure of manually applying patches, written by myself or another person on plain vanilla sources and configure and rebuild a kernel binary manually since I consider that out of focus for this particular writing. Maybe, at some point in the future, I will incorporate this information into here... I assume, that anybody who goes that way is experienced enough to manage to patch, configure and rebuild and further on set up a Linux kernel binary by himself anyway...

The Userland

Now that we have got a Linux kernel with support for dm-crypt, our further needs are the userspace tools to set up and manage dm-crypt virtual block devices. I am going to become `root` and will then issue a simulated run with both, aptitude and apt-get just to show you what it looks like.

```
sa@pc1:~$ su
Password:
pc1:/home/sa# aptitude -s install cryptsetup
Reading package lists... Done
Building dependency tree... Done
Reading extended state information
Initializing package states... Done
Reading task descriptions... Done
Building tag database... Done
The following packages have been kept back:
  python-twisted-web
0 packages upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 0B of archives. After unpacking 0B will be used.
Would download/install/remove packages.
pc1:/home/sa# apt-get -s install cryptsetup
Reading package lists... Done
Building dependency tree... Done
cryptsetup is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
pc1:/home/sa#
```

So, nothing had been installed here but that is perfectly right since `crpytsetup` was already installed. If it would not have been installed the command above should have been `aptitude install cryptsetup` or `apt-get install <package_name>` — no `-s` that is.

Installing a package with Debians package management system is quite comforting since it takes care of all dependencies — the package and its dependencies are getting installed whenever you use a front-end (e.g. `apt-get`) to the package management system to carry out the task.

```
,----[ apt-cache depends cryptsetup ]
| cryptsetup
|   Depends: libc6
|   Depends: libdevmapper1.02.1
|   Depends: libgcrypt11
|   Depends: libpopt0
|   Depends: libuuid1
|   Depends: dmsetup
|   Suggests: udev
|  |Suggests: initramfs-tools
|   Suggests: <linux-initramfs-tool>
|     initramfs-tools
|     yaird
|   Suggests: dosfstools
|   Conflicts: <cryptsetup-luks>
|   Conflicts: hashalot
|   Replaces: <cryptsetup-luks>
`----
```

Use `crpytsetup --help` to get information about the syntax and options the current, LUKS enabled, cryptsetup version features. Currently (July 2007) the cryptsetup version is

```
pc1:/home/sa# cryptsetup --version
cryptsetup 1.0.5
pc1:/home/sa#
```

and the package version

```
sa@pc1:~$ dpl cryptse* | grep ^ii
ii  cryptsetup      2:1.0.5-1      configures encrypted block devices
sa@pc1:~$
```

At a side note, you might have noticed I became the normal user

```
,----[ whoami ]
| sa
```

```
`----
```

again — I issued `exit` on the CLI to change identities (from root to my user name). The reason is simply. From the very first moments with Unix like OSs (Operating Systems) I got used to *"Only become root if absolutely necessary"* and *"Only remain root if absolutely necessary"*. A typo on the CLI as being root could cause devastating results... as normal user the damage would have been at least limited...

Finally, another not essential but nice to have tool is `gnome-volume-manager`. You want to install it onto your system if it is used as a workstation or mobile device which mostly feature a graphical environment.

A tool which I use pretty often but which is not necessarily needed

```
pc1:/home/sa# exit
exit
sa@pc1:~$ apt-file search sfdisk | grep bin/ | cut -d ':' -f1 | xargs dpkg -l | grep ^ii
ii  util-linux                        2.12r-19                          Miscellaneous system utilities
sa@pc1:~$
```

Therefore, you might want to install `util-linux` as well since later on we are going to either use `fdisk` or `sfdisk`. Since both are contained within `util-linux` installing it is fine.

`badblocks` is a tool to check a HDD (Hard Disk Drive) for bad sectors and write random data to the HDD so you might install the package

```
sa@pc1:~$ afs badblocks | grep bin/
e2fsprogs: sbin/badblocks
mtools: usr/bin/mbadblocks
sa@pc1:~$
```

`e2fsprogs` since it contains the tool badblocks as you can see.

Correct Device

At that point, the software we need

- Linux operating system kernel
- Userland tools

should be in place. We can now start to set up dm-crypt on some storage media. Whatever the storage media will be, we need to make first contact so to say. We need to either figure some pci bus id or try it on another, preferably higher level.

Hard Disk Drive - external or internal

In case we would use a HDD (Hard Disk Drive)

```
pc1:/home/sa# fdisk -l | grep ^Disk
Disk /dev/md0 doesn't contain a valid partition table
Disk /dev/hda: 60.0 GB, 60011642880 bytes
Disk /dev/hdb: 60.0 GB, 60011642880 bytes
Disk /dev/md0: 24.9 GB, 24996478976 bytes
Disk /dev/dm-1 doesn't contain a valid partition table
Disk /dev/dm-0 doesn't contain a valid partition table
Disk /dev/sdb: 400.0 GB, 400088457216 bytes
Disk /dev/dm-1: 400.0 GB, 400085283840 bytes
Disk /dev/sdd: 251.0 GB, 251000193024 bytes
Disk /dev/dm-0: 250.9 GB, 250993858560 bytes
pc1:/home/sa#
```

would be a good choice. Using fdisk, one would just have to look at the HDD sizes shown and then find the match to the HDD he wants to use for block-layer encryption. I got two external USB (Universal Serial Bus) HDDs, one 400GB and the other one 250GB.

The screendump above has been made with already mounted HDDs providing block-layer encryption. However, in both cases, after the mounting and with not already mounted HDDs the lines `Disk /dev/sdb: 400.0 GB, 400088457216 bytes` and `Disk /dev/sdd: 251.0 GB, 251000193024 bytes`, both show up. The important information here is the device node — `/dev/sdb` for example. This is the information we need further down to start the setup procedure — it tells me my external 400GB USB HDD is at mount point `/dev/sdb`.

Universal Serial Bus Sticks, Card Readers, iPods...

In case we are not dealing with some HDD (Hard Disk Drive) but some other sort of storage media, we probably need to drill a little bit deeper into the system.

Any device plugged to the computer shows up on the PCI (Peripheral Component Interconnect) bus and gets a unique ID (Identifier) assigned.

PCI Bus Identifier

To take a look at what devices are connected to the PCI bus one might use either `lspci`, `hwdata` or `hwinfo`. Since one could have any kind of special hardware I can just give a common advice how to proceed

1. You know the name of your hardware — manufacturer, vendor, reseller etc. and then the device usually has got a type identifier.

2. With that information you can look up the unique PCI bus ID (Identifier) since any manufacturer got his range of IDs and therefore assigns unique IDs to his devices. There is a file around that lists devices and their PCI IDs. It is called

```
sa@pc1:~$ locate pci.ids
/usr/share/misc/pci.ids
/usr/share/hwdata/pci.ids
sa@pc1:~$
```

and comes with those packages

```
sa@pc1:~$ afs pci.ids
hwdata: usr/share/hwdata/pci.ids
pciutils: usr/share/misc/pci.ids
sa@pc1:~$
```

The most up-to-date `pci.ids` file can be found and downloaded

```
sa@pc1:~$ cd /tmp/
sa@pc1:/tmp$ curl -O http://pciids.sourceforge.net/pci.ids
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  456k  100  456k    0     0   112k      0  0:00:04  0:00:04 --:--:--  148k
sa@pc1:/tmp$ head -n15 pci.ids
#
#       List of PCI IDs
#
#       Maintained by Martin Mares <mj@ucw.cz> and other volunteers from the
#       Linux PCI IDs Project at http://pciids.sf.net/.
#
#       New data are always welcome, especially if accurate. If you have
#       anything to contribute, please follow the instructions at the web site
#       or send a diff -u against the most recent pci.ids to pci-ids@ucw.cz.
#
#       This file can be distributed under either the GNU General Public License
#       (version 2 or higher) or the 3-clause BSD License.
#
#       Daily snapshot on Wed 2007-08-01 01:05:01
#
sa@pc1:/tmp$
```

As you can see, there are daily snapshots — current date is `Wed Aug 1 11:13:21 UTC 2007`. Note, I mostly use `curl` instead of `wget` since it is more powerful but you could also use `wget` to download the file. One might also use

```
pc1:/home/sa# update-pciids
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  119k  100  119k    0     0  63413      0  0:00:01  0:00:01 --:--:-- 93916
Done.
pc1:/home/sa#
```

which is part of the `pciutils` package.

3. Now that you know the PCI ID from `pci.ids`[29] you should be able to find a match with one of the afore mentioned tools (`lspci`, `hwdata` or `hwinfo`).

dmesg

`dmesg` allows to print the kernel ring buffer. The kernel writes constantly to the ring buffer and at some point starts to overwrite the oldest message in the buffer and so forth (principle how a ring buffer works that is).

Therefore, if we plug some storage media to the computer and instantly issue `dmesg`, we might also get good results in order to identify a device

```
sa@pc1:~$ su
Password:
pc1:/home/sa# dmesg | grep -v bound
=UDP SPT=45549 DPT=1027 LEN=486
agpgart: Found an AGP 2.0 compliant device at 0000:00:00.0.
agpgart: Putting AGP V2 device at 0000:00:00.0 into 1x mode
agpgart: Putting AGP V2 device at 0000:01:00.0 into 1x mode
[drm] Loading R200 Microcode
usb 6-4: USB disconnect, address 6
pc1:/home/sa# dmesg | grep -v bound
.39.210 LEN=56 TOS=0x00 PREC=0x00 TTL=122 ID=602 PROTO=UDP SPT=39931 DPT=43719 LEN=36
agpgart: Found an AGP 2.0 compliant device at 0000:00:00.0.
agpgart: Putting AGP V2 device at 0000:00:00.0 into 1x mode
agpgart: Putting AGP V2 device at 0000:01:00.0 into 1x mode
[drm] Loading R200 Microcode
usb 6-4: USB disconnect, address 6
usb 6-3: new high speed USB device using ehci_hcd and address 7
usb 6-3: configuration #1 chosen from 1 choice
scsi6 : SCSI emulation for USB Mass Storage devices
usb-storage: device found at 7
usb-storage: waiting for device to settle before scanning
usb-storage: device scan complete
scsi 6:0:0:0: Direct-Access     Maxtor 6 L250R0           0000 PQ: 0 ANSI: 0
SCSI device sdd: 490234752 512-byte hdwr sectors (251000 MB)
sdd: Write Protect is off
sdd: Mode Sense: 27 00 00 00
sdd: assuming drive cache: write through
SCSI device sdd: 490234752 512-byte hdwr sectors (251000 MB)
sdd: Write Protect is off
sdd: Mode Sense: 27 00 00 00
sdd: assuming drive cache: write through
 sdd: unknown partition table
sd 6:0:0:0: Attached scsi disk sdd
pc1:/home/sa#
```

There are two things to mention regarding the screendump above

- I issued `dmesg` twice after plugging my external USB HDD to the computer. I did so because by the time I did the first invocation, the device has not been already settled so no information could be found in the kernel ring buffer. After waiting a couple of seconds, the second try was much more successful as we can see. The device node that the device had been assigned to is `sdd` resp. `/dev/sdd`.
- `grep -v bound` is just to suppress some other messages that are of non interest in our current case

External/Internal Hard Disk Drive

This section is about setting up block-layer encryption with HDDs (Hard Disk Drives). External (e.g. connected via USB or FireWire) or internal (connected via PATA (Parallel ATA) or SATA (Serial ATA) respectively SAS (Serial Attached SCSI) interfaces).

External Universal Serial Bus Hard Disk Drive

We already know that the external HDD has been assigned to device node `/dev/sdd`. So we got a device node but the HDD is not yet mounted into the file system tree.

Check for Errors and initialize

It is a good idea to check the entire HDD for errors before we start. Not only is this good practise, but also, modern hard disks contain a few *spare sectors*, and if we detect I/O (Input/Output) errors, bad sectors are silently replaced with spare sectors (this is invisible to the OS (Operating System)). So writing and reading the entire disk before we start should allow this mapping from bad to spare sectors to happen.

This step is optional but I strongly recommend to not skip it since it ensures the HDD has got no erroneous sectors plus it writes random data onto the HDD and thus is an AF (Anti-Forensic) countermeasure.

The following command checks the HDD for bad sectors and fills it with pseudo-random data.

```
pc1:/home/sa# badblocks -c 10240 -s -w -t random -v /dev/sdd
```

Be aware of the fact, that it takes around 180 minutes per 100GiB depending on your computers speed (generating random data is very CPU intensive). For a 400GB HDD it will probably take the whole night to complete, so trigger the process before you go to bed or so.

For the more paranoid

This step is also optional. Although, I strongly recommended to not skip the former step, this one is not absolutely necessary. The above example fills the HDD with pseudo-random data, thus the degree of randomness is eventually not enough to withstand an expert forensic attack.

For those of you who would set up a logical virtual block device, with the most possible strength, should carry out another step after the afore mentioned checking for bad sectors and filling with pseudo-random data.

```
pc1:/home/sa# dd if=/dev/urandom of=/dev/sdd
```

This also takes a long time — around 3 minutes per GiB on my 5 years old 32 Bit system.

```
sa@pc1:~$ cat /proc/cpuinfo | grep 'model name'
model name      : Intel(R) Pentium(R) 4 CPU 3.06GHz
sa@pc1:~$
```

For a 400GB (~372 GiB) that takes around

```
sa@pc1:~$ python
Python 2.4.4 (#2, Jul 21 2007, 11:00:24)
[GCC 4.1.3 20070718 (prerelease) (Debian 4.1.2-14)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> (372*3)/60
18
>>>
sa@pc1:~$
```

hours. For that, I used one of my other computers but not my workstation. We can use tools such as `pv` or `bar` in order to have some feedback about the ongoing process and when it is most likely to be finished:

```
sa@sub:~$ type acsn; acsn pv | grep '^pv '
acsn is aliased to `apt-cache search --names-only'
pv - Shell pipeline element to meter data passing through
```

```
sa@sub:/tmp$ acsn bar | grep '^bar '
bar - Show information about a data transfer
```

Now, just to demonstrate it, we will use the file `foobar` rather than `/dev/sdd` and write 100 MiB to it:

```
sa@sub:/tmp$ dd bs=1M count=100 if=/dev/urandom | bar -s 100m -of /tmp/foobar
68+1 records in 8MB/s  eta: 0:00:06  64% [========================================                    ]
68+0 records out
71303168 bytes (71 MB) copied, 11.1042 s, 6.4 MB/s


[ a snapshot at 64% ...]


sa@sub:/tmp$ ll foobar; file foobar
-rw-r--r-- 1 sa sa 100M Jul  4 00:12 foobar
foobar: data
sa@sub:/tmp$
```

As can be seen, block-layer encryption already starts before there is any encryption software (dm-crypt in our case) in place. **It is important to initially fill the storage media with pseudo-random data!**

Partitioning the Hard Disk Drive

Folks are mostly familiar using `fdisk` so I will provide an example for it

```
sa@pc1:~$ su
Password:
pc1:/home/sa# fdisk /dev/sdd

The number of cylinders for this disk is set to 30515.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/sdd: 251.0 GB, 251000193024 bytes
255 heads, 63 sectors/track, 30515 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-30515, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-30515, default 30515):
Using default value 30515

Command (m for help): p

Disk /dev/sdd: 251.0 GB, 251000193024 bytes
255 heads, 63 sectors/track, 30515 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1       30515   245111706   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
pc1:/home/sa#
```

That way I partitioned the HDD to provide a single partition that spans the whole physically available storage space. The user input I provided was as follows (a ordered sequence): `fdisk /dev/sdd RET`, `p RET`, `n RET`, `p RET`, `1 RET`, `1 RET`, `RET`, `p RET`, `w RET`[30].

Next to `fdisk` one might use `cfdisk` or `parted` which are all CLI (Command Line Interface) tools. If there is a personal liking for GUIs (Graphical User Interfaces) one might also use `gparted` for example.

`sfdisk` has become my favorite over the years. I like it especially for the fact that I can use it non-interactive with shell scripts for example. FAI (Fully Automatic Installation), which I like a lot, also makes use of it. It will now follow an example where I made use of `sfdisk` to partition my iPod. I use my iPod (in essence it is nothing but a tiny storage media with a display and a SOC (System-on-a-chip)) not just to listen to music but also as a storage media that I encrypted with dm-crypt[31].

```
 1  pc1:/tmp/our_tmp_dir# sfdisk /dev/sdc << EOF
 2  > ,1,0,*
 3  > ,330,B,*
 4  > ,,83,-
 5  >Checking that no-one is using this disk right now...
 6  OK
 7
 8  Disk /dev/sdc: 497 cylinders, 255 heads, 63 sectors/track
 9  Old situation:
10  Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0
11
12     Device Boot Start     End   #cyls    #blocks   Id  System
13  /dev/sdc1   *      0+      4       5-     40131    0  Empty
14  /dev/sdc2   *      5     496     492    3951990    b  W95 FAT32
15  /dev/sdc3          0       -       0          0    0  Empty
16  /dev/sdc4          0       -       0          0    0  Empty
17  New situation:
18  Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0
19
20     Device Boot Start     End   #cyls    #blocks   Id  System
21  /dev/sdc1   *      0+      0       1-      8032    0  Empty
22  /dev/sdc2   *      1     330     330    2650725    b  W95 FAT32
23  /dev/sdc3        331     496     166    1333395   83  Linux
24  /dev/sdc4          0       -       0          0    0  Empty
25  Warning: more than one primary partition is marked bootable (active)
26  This does not matter for LILO, but the DOS MBR will not boot this disk.
27  Successfully wrote the new partition table
28
29  Re-reading the partition table...
30
31  If you created or changed a DOS partition, /dev/foo7, say, then use dd(1)
32  to zero the first 512 bytes:  dd if=/dev/zero of=/dev/foo7 bs=512 count=1
33  (See fdisk(8).)
```

In lines 1 to 33, we are creating new partitions on the iPods HDD — my iPod runs Linux that is why. Therefor, I use `sfdisk` in script mode throughout line 1 to 4. Once, one understands how it works, it is the fastest and most comfortable way to partition a HDD. `man 8 sfdisk` informs about details. The partition IDs (Identifiers) are given in the fourth field of every line (`0` in line 2, `B` in line 3 and `83` in line 4).

Line 12 to 16 shows the old situation. Line 20 to 24 shows the new situation, which, what is the result of line 1 to 4. If you enter those lines on the CLI (Command Line Interface), you can signal an `EOF` via `C-d` (press `Ctrl + d` at once) in order to signal to sfdisk that you are done with input (I hit `C-d` at the end of line 4).

Below is a listing of the available partition types:

Each partition, regardless if primary or logical partition, can have it is own file system type e.g. `ext3`, `ext2`, `NTFS`, `FAT`, `XFS`, etc. Below, is a table of the types one can choose. Each entry has two columns. The first one is the ID (Identifier) in hexadecimal and the second column is the textual identifier.

```
01 FAT12                       24 NEC DOS                    81 Minix / old Linux          C1 DRDOS/sec
02 XENIX root                  39 Plan 9                     82 Linux swap / Solaris       C4 DRDOS/sec
03 XENIX usr                   3C PartitionMagic recov       83 Linux                      C6 DRDOS/sec
04 FAT16 <32M                  40 Venix 80286                84 OS/2 hidden C: drive       C7 Syrinx
05 Extended                    41 PPC PReP Boot              85 Linux extended             DA Non-FS da
06 FAT16                       42 SFS                        86 NTFS volume set            DB CP/M / CT
07 HPFS/NTFS                   4D QNX4.x                     87 NTFS volume set            DE Dell Util
08 AIX                         4E QNX4.x 2nd part            88 Linux plaintext            DF BootIt
09 AIX bootable                4F QNX4.x 3rd part            8E Linux LVM                  E1 DOS acces
0A OS/2 Boot Manager           50 OnTrack DM                 93 Amoeba                     E3 DOS R/O
0B W95 FAT32                   51 OnTrack DM6 Aux1           94 Amoeba BBT                 E4 SpeedStor
0C W95 FAT32 (LBA)             52 CP/M                       9F BSD/OS                     EB BeOS fs
0E W95 FAT16 (LBA)             53 OnTrack DM6 Aux3           A0 IBM Thinkpad hiberna       EE EFI GPT
0F W95 Ext'd (LBA)             54 OnTrackDM6                 A5 FreeBSD                    EF EFI (FAT-
10 OPUS                        55 EZ-Drive                   A6 OpenBSD                    F0 Linux/PA-
11 Hidden FAT12                56 Golden Bow                 A7 NeXTSTEP                   F1 SpeedStor
12 Compaq diagnostics          5C Priam Edisk                A8 Darwin UFS                 F4 SpeedStor
14 Hidden FAT16 <32M           61 SpeedStor                  A9 NetBSD                     F2 DOS secon
16 Hidden FAT16                63 GNU HURD or SysV           AB Darwin boot                FD Linux rai
```

```
17 Hidden HPFS/NTFS          64 Novell Netware 286       B7 BSDI fs                   FE LANstep
18 AST SmartSleep            65 Novell Netware 386       B8 BSDI swap                 FF BBT
1B Hidden W95 FAT32          70 DiskSecure Multi-Boo     BB Boot Wizard hidden
1C Hidden W95 FAT32 (LBA)    75 PC/IX                    BE Solaris boot
1E Hidden W95 FAT16 (LBA)    80 Old Minix                BF Solaris
```

Creation of the logical virtual block device

We are now ready to create the logical virtual block device also known as *dm-crypt device-mapper device*.

dm-crypt works by transparently translating[32] (within the kernel) between a physical block-layer partition (which is encrypted) and a logical partition (the logical virtual block device) which we can then mount and use as usual.

- The physical (encrypted, locked) partition will be `/dev/<a_name>`
- The logical (unencrypted, unlocked) partition will be `/dev/mapper/<a_name>`

We know my external USB (Universal Serial Bus) HDD is at `/dev/sdd` so I might issue

```
pc1:/home/sa# cryptsetup --verbose -c aes-cbc-essiv:sha256 --verify-passphrase luksFormat /dev/sdd1

WARNING!
========
This will overwrite data on /dev/sdd1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
```

in order to format (`luksFormat`) i.e. set up the logical virtual block device. As mentioned before, one might issue `cryptsetup --help` to figure the syntax to cryptsetup and also to see what switches etc. are available.

Notice the `-c aes-cbc-essiv:sha256`. This is what I already mentioned above. In addition to that, we use verbose mode (`--verbose`) and the `--verify-passphrase` switch which prompts us twice for the password instead of just once which might lead to non-detected typos, meaning you become the one unsuccessfully launching cracking attempts on your own storage media.

If you prefer to use a key-file, you might study the possibilities reading the manual page (`man 8 cryptsetup`) and/or issue the afore mentioned `cryptsetup --help` and come up with something like that

```
pc1:/home/sa# cryptsetup -v --cipher serpent-cbc-essiv:sha256 --key-size 256 luksFormat /dev/sdd1 <key
```

The alert reader might have noticed that I used `--verbose` once and for the other command sequence I used `-v`. Those are synonymous. The first one is called a *long option* and was introduced by the GNU (GNU is Not Unix) project, the latter one is called a *short option*. The advantage of long options is their high expressiveness, the downside is there is more to type.

The logical virtual block device is then ready to be opened[33]. After opening it by providing the correct password (line 2), I decided to take a look (`ls -l /dev/mapper/`) at all mapper devices which are opened at that time. Line 8 just shows the control file used by dm-crypt for internal handling of logical virtual block devices.

```
 1  pc1:/home/sa# cryptsetup luksOpen /dev/sdd1 kim
 2  Enter LUKS passphrase:
 3  key slot 0 unlocked.
 4  Command successful.
 5  pc1:/home/sa# ls -l /dev/mapper/
 6  total 0
 7  brw-rw---- 1 root disk 253,  1 2007-07-31 14:49 alan
 8  crw-rw---- 1 root root  10, 63 2007-07-31 12:01 control
 9  brw-rw---- 1 root disk 253,  0 2007-07-31 14:48 kim
10  pc1:/home/sa#
```

As can be seen, in line 1 I issued the command to open the encrypted HDD and therefore it is now providing the logical virtual block device `kim` (line 9).

Create a File System on top of the Logical Virtual Block Device

The next step is to create a file system on top of the logical virtual block device. This is just like making a normal file system, we need to point `mkfs` at the former created logical virtual block device (i.e. `/dev/mapper/kim` in our case[34]). We can use any file system with any options applicable... just as usual without using block-layer encryption.

I use Ext3 respectively Ext4

```
pc1:/home/sa# mkfs.ext3 -j -m 1 -O dir_index,filetype,sparse_super /dev/mapper/kim
```

but do have a fond for XFS

```
pc1:/home/sa# mkfs.xfs /dev/mapper/kim
```

you might also download the tools to manage a XFS file system

```
,----[ dpkg -l xfs* | grep ^ii ]
| ii  xfsdump        2.2.45-1       Administrative utilities for the XFS filesys
| ii  xfsprogs       2.9.0-1        Utilities for managing the XFS filesystem
`----
```

As I said, one might use <u>any</u> file system with <u>all</u> provided options — nothing differs from the usual case of creating a file system on storage media providing plain I/O (Input/Output) without any kind of block-layer encryption.

Mount the dm-crypt Device for Usage

This is just like a normal mount, except we use the logical (`/dev/mapper/<a_name>`) device that can be mounted into the file system tree (which is a B-tree like structure) as usual and thus providing transparent I/O (Input/Output) to the HDD (Hard Disk Drive).

```
 1  pc1:/home/sa# fdisk -l | grep ^Disk
 2  Disk /dev/md0 doesn't contain a valid partition table
 3  Disk /dev/dm-0 doesn't contain a valid partition table
 4  Disk /dev/dm-1 doesn't contain a valid partition table
 5  Disk /dev/hda: 60.0 GB, 60011642880 bytes
 6  Disk /dev/hdb: 60.0 GB, 60011642880 bytes
 7  Disk /dev/md0: 24.9 GB, 24996478976 bytes
 8  Disk /dev/sda: 251.0 GB, 251000193024 bytes
 9  Disk /dev/dm-0: 250.9 GB, 250993858560 bytes
10  Disk /dev/sdd: 400.0 GB, 400088457216 bytes
11  Disk /dev/dm-1: 400.0 GB, 400085283840 bytes
12  pc1:/home/sa# cryptsetup luksOpen /dev/sda1 kim
13  Enter LUKS passphrase:
14  key slot 0 unlocked.
15  Command successful.
16  pc1:/home/sa# mount /dev/mapper/kim /media/usb0
17  pc1:/home/sa# exit
18  exit
19  sa@pc1:~$
```

The only thing to note here is, that with `gnome-volume-manager` installed, after you provided the correct password, a window pops up (the file manager) showing that the disk has already been mounted to something like `/media/disk` or so. That is fine since normally one does not have to issue another `mount` (line 16) as I did. The reason I issued another mount is because I also wanted to have the disk mounted to `/media/usb0` for various settings on my system that depend on that path (udev rules would also work).

The `di -h` below shows the result of the former mount procedure. The fact that `/dev/sda` is used instead of `/dev/sdd` is just because those parts have been written on different days and so meanwhile the HDDs had been switched off and got a new (different) device node assigned by the Linux kernel. One might use udev rules to get persistent device node namings if he wants to...

```
,----[ di -h | head -n1 && di -h | grep media ]
| File System         Mount               Size     Used    Avail %Used fs Type
| /dev/dm-0          /media/disk       233.6G   140.5G    93.1G  60%  xfs
| /dev/dm-1          /media/disk-1     372.5G   340.4G    32.1G  91%  xfs
| /dev/mapper/kim    /media/usb0       233.6G   140.5G    93.1G  60%  xfs
| /dev/mapper/alan   /media/usb1       372.5G   340.4G    32.1G  91%  xfs
`----
```

As we can see, one disk and two mount points. Also, note the size difference reported by `di` and `fdisk` (e.g. 400GB vs 372.5GiB).

We are done. Now we have another 233GiB (~250GB) of storage media at our hands to keep was is a secret a secret and thus maintaining our self-determination, securing our business, the governmental agency can fulfill imposts it has to obey to or we can equip the UMPC (Ultra-Mobile PC) the brave soldiers carry around on the 21st century battlefield with an encrypted HDD (Hard Disk Drive) or SSD (Solid State Drive).

Unmount and close the dm-crypt Device

After using the logical virtual block device we need to unmount and close the device again. It is important to close the device in order to avoid the following

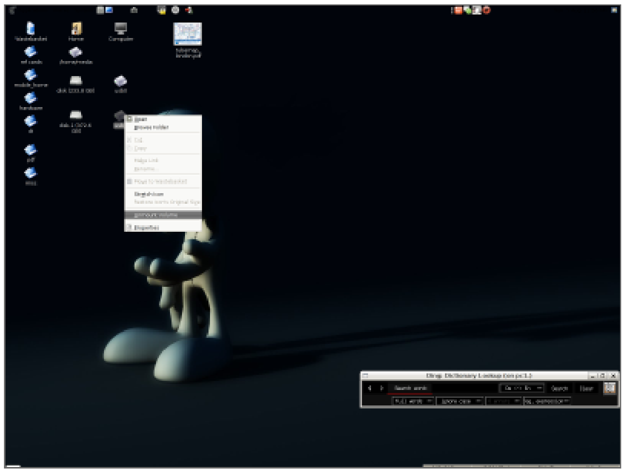
The visitors and the incautious mind

> They came, and although I managed to unmount my encrypted HDD before I left the house, they where able to reveal all the secrets stored onto the HDD simply because the device was still open. All they had to do was to mount the still open device e.g. `/dev/mapper/alan` again and then access the data.

The below screendump shows, both, `/dev/mapper/kim` and `/dev/mapper/alan` are open simply because they show up when we issue `ls -l /dev/mapper`.

```
pc1:/home/sa# ls -l /dev/mapper/
total 0
brw-rw---- 1 root disk 253,  1 2007-08-03 05:55 alan
crw-rw---- 1 root root  10, 63 2007-08-03 05:42 control
brw-rw---- 1 root disk 253,  0 2007-08-03 05:51 kim
pc1:/home/sa#
```

So, the device needs to be closed. Before closing it, it must be unmounted which can be done via the GUI (Graphical User Interface).



Note the two different icons per HDD. I noted above that I mounted the HDD to a second mount point — so there they are as can be seen. When I want to unmount a HDD, I have to click both icons per HDD and use the menu item `Unmount Volume` twice.

The second choice next to the GUI is to use the CLI (Command Line Interface). Again, please note that I had chosen to mount the logical virtual block device twice thus the double umount.

```
pc1:/home/sa# umount /media/usb0
pc1:/home/sa# umount /media/disk
pc1:/home/sa#
```

Finally, the current status is that both former devices (`/dev/mapper/kim` and `/dev/mapper/alan`) are open, with `/dev/mapper/kim` already unmounted as we know. Then, one (`kim`) out of two is closed — **this is the only way (by unmounting AND subsequently closing a device) to ensure that secrets are not getting revealed.**

```
pc1:/home/sa# ls -l /dev/mapper/
total 0
brw-rw---- 1 root disk 253,  1 2007-08-03 05:55 alan
crw-rw---- 1 root root  10, 63 2007-08-03 05:42 control
brw-rw---- 1 root disk 253,  0 2007-08-03 05:51 kim
pc1:/home/sa# cryptsetup luksClose kim
pc1:/home/sa# ls -l /dev/mapper/
total 0
brw-rw---- 1 root disk 253,  1 2007-08-03 05:55 alan
crw-rw---- 1 root root  10, 63 2007-08-03 05:42 control
pc1:/home/sa#
```

Logical Volume Manager

For now we have been talking about *static* storage sizes which had then been encrypted. What if it becomes necessary to resize (grow/shrink) the file system and its underlying logical virtual block device?

No problem. We are going to put another layer below the whole block-layer encrypted shebang. This additional layer is called LVM (Logical Volume Manager) and is the Linux implementation providing us with storage virtualization. This whole subject (block-layer encryption on top LVM) is covered on my dedicated LVM page.

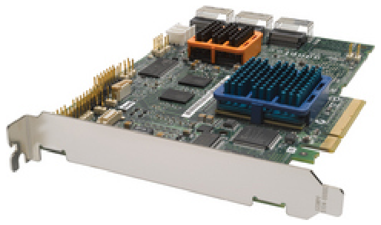
I strongly recommend for using block-layer encryption on top of LVM for all practical purposes! Why this combination is generally considered best-practice can be read on my dedicated LVM page and on this page respectively. In short, LVM is needed to provide a flexible storage layer and block-layer encryption to keep a secrets and protect assets from others.

This combined subject (block-layer encryption + LVM) may sound complex and huge at first glance but then, I am providing very good information I think — even the novice can deal with easy-to-ape information as I am used to write this whole website/platform. So, anybody go for block-layer encryption atop LVM now! ;-]

Hardware RAID

In this section, I am going to show how to use dm-crypt and LUKS to encrypt some hardware RAID array which is managed with Adaptec's 31205 PCIe (PCI Express) RAID HBA (Host Bus Adapter) controller. This PCIe controller can manage up to 12 SATA/SAS (Serial Attached SCSI) HDDs whereas one might mix SATA and SAS HDDs although it is recommended not to do so for performance reasons and the like. I am just using SATA HDDs in this server.

I also strongly recommend the use of LVM (Logical Volume Manager) at a lower level in order to do online expansion etc. of some hardware RAID array that one intends to fully encrypt. We can then build our logical encrypted block device atop (see above) the LVM layer. In fact, I make use of LVM all the time — even for external USB HDDs.



Hardware RAID + LVM + Block-layer Encryption

There are two different ways to do this — the straight forward one and the more flexible but also more complicated one.

Installation from Scratch

We have a computer (e.g. workstation) into which we installed a RAID HBA (Host Bus Adapter) card with some HDDs connected to it. There is no OS (Operating System) installed so far, just the hardware is assembled and ready to go. Next we initialize the RAID array, make it bootable and install DebianGNU/Linux onto that RAID array, all as usual.

The actual work i.e. setting up block-layer encryption with LVM on top of our hardware RAID is done by the Debian installer. Yes, it really is that easy! All we need to do after we have initialized and made the RAID array bootable is to for example use a netinstall image and install Debian. Finally, we end up with a setup as mentioned in example 4 from above.

OS already installed

The second case is where we already have an up and running system i.e. some computer with Debian installed plus there is also a RAID HBA card installed into the system. In this case, there are two possible ways to proceed depending on the yet existing setup.

1. The OS is installed onto some RAID array that belongs to the same RAID HBA card which we want to use for the RAID array that we are going to use with LVM and dm-crypt/LUKS or
2. the OS is not installed onto some RAID array the belongs to the same RAID HBA card. In this case, the OS might be installed onto a RAID array onto some other controller or some other block device e.g. a single HDD, SSD (Solid State Drive), etc.

Actually, for us it does not matter which case our yet existing setup is because from now on, the procedure is the same for both. However, now we face the choice between putting the

- encryption layer on top of the LVM layer (example 2) or putting the
- LVM layer on top of the encryption layer (example 4)

Both have their particular weaknesses and strengths as I stated here. Bellow follows a listing of all the necessary steps (in chronological order) needed to accomplish our goal which is to use a RAID array with LVM and dm-crypt/LUKS.

- Assuming that we have dealt with the hardware part already i.e. connecting the needed HDDs to our RAID HBA card, now is the time to initialized the HDDs in the array and create the RAID array which includes opting for a particular RAID level such as RAID 6 for example. For this, I use the ASM (Adaptec Storage Manager), assuming that we use an Adaptec RAID HBA card.

  - Next, we figure out the device-path to our RAID array by, for example, using `fdisk` as described above.
  - Now is the time where we decide if we either put the LVM layer on top of the encryption layer or vice versa; once a decision is made we proceed as usual i.e. as the LVM and dm-crypt pages (current one) explain it.

It is always the same, we pick a block device (HDD, USB stick, RAID array, LUN (Logical Unit Number), etc.) put our layers (storage e.g. LVM, encryption e.g. dm-crypt, backup e.g. DRBD, etc.) in some particular order onto it. Finally we put a file system on top of the whole stack and be done.

Growing the Storage

If we wanted to grow/shrink the now existing storage, we can easily do so — it does not matter if we use a hardware RAID or just the ordinary single HDD setup.

Solid State Drive

Of course there are differences between SSDs (Solid State Drives) and HDDs (Hard Disk Drives) but not so much with regard to block-layer encryption. SSDs are completely different in design to HDDs. However, both can be used in the same way (as a storage media connected to the same I/O (Input/Output) interface type e.g. SATA (Serial ATA)). Because they can be used in the same way and thus become quite transparent to the OS (Operating System) (roughly speaking) the following is true.

Secure deletion of data stored onto an SSD (Solid State Drive)

Securely deleting with tools like `shred` etc. will not work either i.e. forensics might be able to restore data (i.e. keys, passwords, etc.) on unencrypted SSDs because SSD hardware (the controller) uses a technique called wear levelling. The countermeasures are the same as with AF (Anti-Forensic) for HDDs. There are SSDs out there (reportedly the US-AirForce and US-Navy uses them) which can be forced to temporary disable wear levelling in order to securely erase flash cells on SSDs and thus securely erase/destroy information on unencrypted SSDs. However, as of now (August 2007), those SSDs are insanely expensive ($40+ per GiB). The good news is, with dm-crypt also customer SSDs turn into secure storage media devices in sense of securely destroying information.

The bottom line is, neither with HDDs (Hard Disk Drives) nor SSDs (Solid State Drives) need we fear our secrets may be revealed.... as long as we use block-layer encryption.

USB Stick

This works exactly the same as to what I already mentioned above. The only difference is that now we deal with some USB stick instead of some HDD.

Swap Partition

The easiest way to accomplish an encrypted swap partition is when we use the Debian installer and choose to have and encrypted swap partition. The installer does all the magic and sets it up in the process of installing Debian.

On the other hand, should we already have an up and running system with Debian installed but our swap space is not encrypted yet, then we need to set up an encrypted swap space manually. This can be done in two ways:

- We either use the storage space already in use by the swap space and set up block-layer encryption with dm-crypt and LUKS so we end up with and encrypted swap space that resides in the exact same place as the non-encrypted swap space did before, or
- we use some new storage space to set up our encrypted swap space

Either ways, it does not matter what kind of block device we are going to use respectively is already in use by the swap space right now — a separate HDD, RAID array, partition from some HDD already in use, LV, etc.

For both cases we need to identify the storage space i.e. its device path like for example `/dev/sdb3` for the third partition on the second HDD. In case we decide to use the current swap space and *transform* it into an encrypted one using dm-crypt and LUKS, one should use `swapoff`, `mkswap` and `swapon` respectively.

Putting the encryption layer onto the block device earmarked to hold the swap space is the same procedure for both cases i.e. the one already shown above. Once this is done the only thing left to do is automatize things i.e. make entries (respectively alter the old one(s)) in `/etc/fstab` and `/etc/crypttab`. Go here for more information.

Automatize

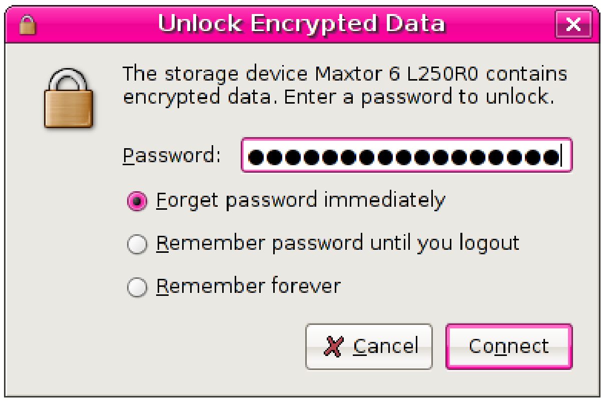
What I showed above should be avoided if anyhow possible since it is a tedious and ever repeating task. Instead we want to just provide the password and not further intact with the computer during the booting phase. This sections shows how one might speed up things when using block-layer encryption.

Graphical User Interface

I use GNOME (GNU Network Object Model Environment) so I have

```
sa@pc1:~$ dpl gnome-volume* | grep ^ii
ii  gnome-volume-manager 2.17.0-2      GNOME daemon to auto-mount and manage media devices
sa@pc1:~$
```

installed. `dpl` is just an alias in my .bashrc. With `gnome-volume-manager` installed, after I plug the USB cable to my computer or when already plugged, switching on the external HDD (Hard Disk Drive), I am presented with the below GUI. I can then enter the password and decide to store it or not.



The storage device Maxtor 6 L250R0 contains encrypted data. Enter a password to unlock.

Password: ●●●●●●●●●●●●●●●●●●

[✓ Forget password immediately]
Remember password until you logout
Remember forever

Using gnome-volume-manager mounting and providing the password all gets very easy

After I entered my password (I always do a copy paste from a password manager application or an encrypted file containing my passwords and other sensitive information),

```
,----[ cat ~/.sec/README ]
| Usage:
```

```
|
| - `gpg --symmetric <file>' to encrypt a file and
| - `gpg --decrypt <file>.gpg' to decrypt it
`----
```

the HDD got mounted as `luks_crypto_eeeeedc3-6627-4d28-bbb4-d7f31924ba38` as you can see.

```
sa@pc1:~$ ls -l /dev/mapper/
total 0
brw-rw---- 1 root disk 253,  0 2007-07-05 11:15 alan
crw-rw---- 1 root root  10, 63 2007-07-05 11:02 control
brw-rw---- 1 root disk 253,  1 2007-07-06 22:33 luks_crypto_eeeeedc3-6627-4d28-bbb4-d7f31924ba38
sa@pc1:~$ ls -1 /media/ | grep ^disk
disk
disk-1
sa@pc1:~$ ls -l /media/disk-1
total 0
drwxr-xr-x 3 sa sa 29 2006-07-11 00:33 xxxxxxxxxxxxxxxxxx
drwxr-xr-x 4 sa sa 29 2007-05-28 14:21 xxxxxxxxxxxxxxxxxx
drwxr-xr-x 8 sa sa 29 2006-05-22 07:39 xxxxxxxxxxxxxxxxxx
drwxr-xr-x 3 sa sa 29 2006-11-22 05:02 xxxxxxxxxxxxxxxxxx
drwxr-xr-x 4 sa sa 30 2007-05-29 03:28 xxxxxxxxxxxxxxxxxx
sa@pc1:~$
```

I can now access all data as usual as the screendump above shows (remember what I said about that dm-crypt is transparent encryption). Note that the content of `/media/disk-1` has been obfuscated to `xxxxxxxxxxxxxxxxxx` since I consider that a secret.

Command Line Interface

There are a few CLI tools and files that can be used to automatize things e.g. automatically mount encrypted block devices and prompt the user for their passwords while doing so.

What is also important is not just what each of them does but how they interact with each another. The reader might take a look at the man pages (`man 5 crypttab`, `man 5 fstab`) and files (`/etc/init.d/cryptdisks`) which explain it in detail.

Miscellaneous

This section is used as a collection for information not directly related to dm-crypt but nevertheless has relations to block-layer encryption.

HDDs providing hardware builtin block-layer encryption

That is a conventional HDD (Hard Disk Drive) which is equipped with hardware which provides encryption. As of now (June 2007) such disks are not widespread but we will probably see a rise in the near future as the market demands it more than it does now. My guess is that as of now *normal* people are simple not aware enough of the issue having non encrypted data on their disks... However, there are some vendors selling such HDDs especially targeted at the 2.5" market because those disk are used in notebooks.

- Seagate momentus FDE and its marketing-level like explanation
- Hitachi Travelstar
- no encryption but sort of self-destruction feature for HDDs; quite interesting ;-]
- etc.

1. A geek is an individual who is fascinated by knowledge and imagination, usually electronic or virtual in nature. *(Wikipedia)*
2. This term is misinterpreted almost all the time.
3. A group of people, which can be as small as one person.

4. For all of them, may they act as a matter of insanity or well though out plans based on common logic it doesn't matter for the one being the chosen victim.

5. Don't get me wrong here. I think America (same goes for their allies) is a great country (in fact I love it a lot) with great people living there but... Let me do a quote: *Fighting fire with fire only gets you ashes! — Abigail van Buren (1918 -)* and another one: *I like to believe that people in the long run are going to do more to promote peace than our governments. Indeed, I think that people want peace so much that one of these days governments had better get out of the way and let them have it. — Dwight D. Eisenhower*

6. I don't know whether to fear a bunch of ragtag freedom fighters from another country, or my own government's 1984-style policies. We are waging a war on terror and we don't even know who the fuck the real enemy is. *(The Urban Dictionary)*

7. At this point information still is a secret. People, businesses etc. are willing to pay in order to know secrets. This is the moment where even children should sense the value of being in power *of who knows what at what time to what conditions*.

8. The data could be read by a criminal but not be interpreted/decrypted. So, loss of a computers hard disk that holds encrypted data does not cause harm even if lost to criminals — it is simply worthless for anybody without the key to decrypt the data stored on the disk. That, of course goes for any other device as well: CD-ROM, iPod, DVD, mobile phone, USB stick, floppy Disk etc. In fact every device which is able to hold data can hold encrypted data.

9. Note, that chaos does not equal anarchy — most people have a completely wrong understanding of the term anarchy. Anarchy is not the absence of social order within a society but only the absence of some sort of governmental powers that enforces that order. In fact, anarchy would be preferable to democracy since humans would live in peace and harmony without the need of powers that enforce that order, but that would in turn require a better human species. Since that is not the case, best we came up with until now is *democracy* (nowadays this is surely another democracy (better for rulers, worse for the people) than the old Greeks had back then...) — certainly not the best thing out there but as it seems the best working thing to keep folks from fighting each another on a daily basis since some ruling powers created an aggressive environment in a social, political and economical way. *Do not raise your eybrowne but better get informed what press and media is telling you wrong since decades about the term anarchy and furthermore rethink democracy — what it is and what it probably should be.*

10. It seems the human race cannot be without that rulers <—> people thingy. Some say *what is best* in this regards is not the best for the majority they rule but for the rulers themselves — not always is what is best for the rulers also the best for the people.

11. You can use http://www.goodpassword.com/index.htm to get a good one. Use at least one that consists of 13 or more characters.

12. This image is a screenshot I took from my desktop.

13. Take a look at the architecture of EVMS.

14. Determines if it is possible to do I/O (Input/Output) on a storage (i.e. VG in our case) while the VG is moved around, resized etc. (= online) or if I/O is not possible during that time (= offline).

15. The task of creating a partition. It is the very first thing one does. After partitioning, what follows is mostly the creation of a file system *on top* of the partition. After that the partition is mounted and can be used to store data.

16. The device-mapper within the linux kernel providing encryption for block devices.

17. LUKS is a standard describing encryption in conjunction with dm-crypt but no actual tool to use. The tool which finally provides the implementation of LUKS and can be practically used is called `cryptsetup` — a userspace CLI (Command Line Interface) tool that comes prepackaged with DebianGNU/Linux.

18. A specialized device for use on a network. For example, Web servers, cache servers and file servers can be implemented as general-purpose computers with the appropriate software or as network appliances, which are computers dedicated to a single function and cannot do anything else. See server appliance, Internet appliance and Web cache. *(The Free Computing Dictionary)*

19. A hash can be used as key next to many other applications.

20. Of course no one would name it that way since that would be to obvious — less suspicious would be `/dev/mapper/mynastygirlfriend` since the curious folks could load such stuff from the Internet in barrels.

21. Note, that what might look as a random *thing* can be well described with mathematics e.g. stochastic theory and probabilistic theory and therefore it can be said that even if used on a daily basis by humans it is believed that true randomness only exists with quantum physics i.e. if for example random boy gets married to random girl after the *random* event of love at first sight, that is for sure no randomness but can be well described with stochastic theory and the like.

22. The following procedure (attack) is also known as dictionary attack.

23. mapping - a function such that for every element of one set there is a unique element of another set. *(The Free Dictionary)* A modern hard drive comes with many spare sectors. When a sector is found to be bad by the firmware of a disk controller, the disk controller remaps the logical sector to a different physical sector. *(Wikipedia)*

24. For now we can be satisfied with the rough assumption of: ballooning factor = number of physical sectors onto the HDD.

25. Please note, this section has partially been taken from (Wikipedia) (text and images) but has been tailored towards dm-crypt.

26. http://grouper.ieee.org/groups/1619/email/msg00253.html and http://lkml.org/lkml/2005/1/24/54

27. Later, we will hear how an IV (Initialization Vector) comes into play as well for each block/sector on the encrypted virtual block device, sitting on top of an HDD (Hard Disk Drive) for example.

28. The intention with block-layer encryption is not only to hide WHAT you have stored onto some storage media but also to hide THAT you have stored some particular data. In simple words, if you store a document/mp3/movie/etc. that you are not allowed to posses onto your storage media (HDD, USB stick, DVD, etc.), it can be proven (without the need to decrypt) that you stored that stuff. Denying you did not copy/download etc. this or that does not work — the simple fact that you posses it now is enough. Any document can be prepared with a redundant piece of code (a watermark), leaving intact the mp3/movie/.doc/PDF/etc. but revealing that you stored it even that you are using block-layer encryption. You have to use block-layer encryption with additional parameters like ESSIV (Encrypted Salt-Sector Initialization Vector) for example in order to be secure.

29. Unfortunately not all manufactures provide information to the public about their device to ID mapping thus the file is as good as the information a manufacturer provides.

30. Note that `RET` is what Emacs users might be familiar with — it just tells to push the return key at this point.

31. However, this iPod is about four years old — today I would not buy a proprietary thing anymore because of the emerging FOSS (Free and Open Source Software) alternatives plus I think Apple sucks a bit since they sells their stuff completely overpriced ... but then that is another story, also known as Apple Hype.

32. HDD (Hard Disk Drive) I/O (Input/Output) is happening as usual from the point of view of any userspace application.

33. Opening in this regard means to unlock the device for subsequent en/decryption to/from the HDD (Hard Disk Drive) respectively any other storage media that has been equipped with dm-crypt.

34. This has to be the just created logical virtual block device i.e. `/dev/mapper/kim` but not the underlying `/dev/sdd`. `/dev/sdd` is the encrypted block device e.g. a HDD (Hard Disk Drive) of USB (Universal Serial Bus) stick etc.