GBDE - GEOM Based Disk Encryption

Poul-Henning Kamp
The FreeBSD Project
phk@FreeBSD.org

Abstract

The ever increasing mobility of computers has made protection of data on digital storage media an important requirement in a number of applications and situations. GBDE is a strong cryptographic facility for denying unauthorised access to data stored on a ‘‘cold’’ disk for decades and longer. GBDE operates on the disk(-partition) level allowing any type of file system or database to be protected. A significant focus has been put on the practical aspects in order to make it possible to deploy GBDE in the real world. [1]

1. Losing data left and right

In the last couple of years, gentlemen of the press have repeatedly been able to expose how laptop computers containing highly sensitive or very valuable information have been lost to carelessness, theft and in some cases espionage. [THEREG]

The scope of the problem is very hard to gauge, since it is not a subject which the involved persons and, in particular, institutions are at all keen on having exposed. However, a few data points have been uncovered, revealing that the U.S. Federal Bureau of Investigation loses, on average, one laptop every three days. [DOJ0227]

When a computer is lost, stolen or misplaced, it is very often the case that the computer hardware represents a value which is insignificant compared to the value of the disk contents. More often than not, the only reason the press heard about it was that the material on the disk was ‘‘hot’’ enough to make the loss of control rattle people at government level.

While it is easy to blame these incidents on ‘‘user error’’, as is generally done, doing so makes it a very hard problem to fix. Human nature being what it is, seems to remain just that.

In the absence of technical counter measures, administrative measures have been applied, generally with abysmal results. In one case, a bureaucracy has handled the problem according to what could easily be mistaken for the plot from a classic Buster Keaton movie:

> First a laptop was forgotten and lost in a taxi-cab. New policy: always drive your own car if you bring your laptop. Then a car was stolen, including the laptop in the trunk. New policy: always bring your laptop with you. The next laptop was stolen from a pub while the owner was bowing to the pressures of nature. New policy: employees are not to carry their own laptops outside the office at any time. Laptops will be transported from and to the employees home address by the agency security force and will be chained and locked to a ring in the wall installed by the company janitors. All requests must be filed 3 days in advance on form ##-#. [PRIV]

2. Protecting disk contents

Protecting the contents of a computer’s disk can in practice be done in two ways: by physically securing the disk or by encrypting its contents.

Physical protection is increasingly impossible to implement. It used to be that disk drives could only be moved by forklift, but these days a gigabyte disk is the size, but not quite yet the thickness, of a postage stamp. While computers can be tied down with wires and bars can be put in front of windows, such measures are generally not acceptable, or at least not judged economically justified in any but the most sensitive operations.

That leaves encryption of the disk contents as the only practical and viable mode of protection, and both the practicality and the viability has been somewhat in doubt.

Until recently, nearly all aspects of cryptography were a highly political issue, this has eased a lot in the last couple of years and there now ‘‘only’’ remain a number

[1] This software was developed for the FreeBSD Project by Poul-Henning Kamp and NAI Labs, the Security Research Division of Network Associates, Inc. under DARPA/SPAWAR contract N66001-01-C-8035 (‘‘CBOSS’’), as part of the DARPA CHATS research program.

of rather fundamental questions in the area of law enforcement and human rights, which are still unsettled.

With the political issues mostly out of the way, the next roadblock is practical: While use of cryptography can never be entirely transparent, the overhead and workload it brings must be reasonable.

2.1. Application level encryption

Encryption at the application level has been available for a number of years, primarily in the form of the PGP [PGP] program. This is about as intrusive and demanding as things can get: the user is explicitly responsible for doing both encryption and decryption and must enter the pass-phrase for every operation. [2]

Apart from the inconvenience of this extra workload, many organisations would trust their users neither to get this right nor even to want to get it right. From an institutional point of view it is important that cryptographic data protection can be made mandatory.

2.2. Filesystem level encryption

Encryption at the file system level is a tried and acknowledged method of providing protection, but it suffers from a number of drawbacks, mainly because no mainstream file systems offer encryption.

Encrypting file systems are speciality items, which means increased cost and system administration problems of all sorts.

And since practically all operating systems use their own file system format, cross platform fully functional file systems are very rare. This means that a typical organisation will have to operate with a handful of different methods of encryption, which translates to system administration overhead, user confusion and extra effort to pass security and ISO9000 audits.

A secondary, but increasingly important issue is that data which are stored in databases on raw disk, operating system paging areas and other such data are not protected by a cryptographic file system. To protect these would mean adding yet another set of encryption methods, which leads to a situation which is very hard to handle practically and administratively.

Finally, file systems have a complex programming interface to the operating system, which traditionally has been subject to both version skew and compatibility problems.

[2] Interestingly, this is so impractical in real world use that various applications with PGP support resort to caching the pass-phrase at the application level, thereby weakening the protection a fair bit.

2.3. Disk level encryption

Encryption at the disk level can protect all data, no matter how they are stored, file system, database or otherwise.

To a user, encryption at the disk level would require authentication before the computer can be used, everything functioning transparently thereafter, with all disk content automatically protected.

Given that the programming interface for a disk device is very simple and practically identical between operating systems, there are no technical reasons why the same implementation could not be used across several operating systems.

All in all this is a close to ideal solution from an operational point of view.

There are significant implementation issues however. In difference from the higher levels, encryption at the disk level has no way of knowing a priori which sectors contain data and which sectors do not; neither is knowledge available about access patterns or relationships between individual sectors.

Where application level or file system based encryption schemes can key each file individually, a disk based encryption must key each and every sector individually, even if it is not currently used to hold data.

It has been argued that the encryption ideally should happen in the disk-drive, and while there are steps in this direction, they do unfortunately seem to have been made for the wrong reasons by the wrong people [CPRM], and have consequently not gained acceptance.

Provided the owner of the computer remains in control of the encryption, I see no reason why encryption in the disk drives should not gain acceptance in the future.

3. Why this is not quite simple

Several implementations have been produced which implement a disk encryption feature by running the user provided passphrase through a good quality one-way hash function and used the output as a key to encrypt all the sectors using a standard block cipher in CBC mode. A per sector IV for the encryption is typically derived from the passphrase and sector address using a one-way hash function. Two typical examples are [CGD] and [LOOPAES].

Unfortunately this approach suffers from a number of significant drawbacks, both in terms of cryptographic

strength and deployability.

For data to stay protected for decades or even lifetimes, sufficient margin must exist not only for technological advances in brute force technology, but also for theoretical advances in cryptoanalytical attacks on the algorithms used.

Protecting a modern disk, typically having a few hundred millions of sectors, with the same single 128 or 256 bits of key material offers an incredibly large amount of data for statistical, differential or probabilistic attacks in the future.

Worse, because the sectors contain file system or database data and meta data which are optimised for speed, the plaintext sector data typically have both a high degree of structure and a high predictability, offering ample opportunities for statistical and known plaintext attacks.

This author would certainly not trust data so protected to be kept secret for more than maybe five or ten years against a determined attacker.

But far more damning to this method is that there can only be one single passphrase for the disk. This effectively rules out the ability for an organisation to implement any kind of per-user or multilevel key management scheme: the only possible scheme is ‘‘one key per disk’’.

Add to this that to change the passphrase the entire disk would have to be decrypted and re-encrypted, and we have a model which may work in theory, and can be made to work in practice for a determined individual, but which would fast become an operational liability for any organisation.

4. Designing GBDE

The initial design phase of GBDE focused on determining a set of features which would make it both possible and practical for people and organisations to deploy GBDE in routine use.

The first decision has already been described and argued for: Encryption must happen at the sector or ‘‘raw disk’’ level, in order to be comprehensive, universal and portable.

The second decision was dictated by the fact that all security policies we have ever seen, contain a rule which says ‘‘passwords must be changed every N {days,weeks,months}’’. This is sound thinking, and GBDE should support it. While changing the passphrase does not necessarily have to be instantaneous, decrypting and re-encrypting an entire disk would likely take more than a day with currently available hardware, and is consequently out of the question.

The third design criterion came from the fact that people forget passphrases, and while loosing the entire content of the disk as punishment could be seen as a large-calibered educational device, it is not acceptable from a real world perspective: there must be some kind of multiple access paths.

Given that GBDE is open source software, there is little more than symbolic value in a hierarchical set of passphrases: changing the source code to bypass the hierarchy cannot trivially be prevented. It is also not clear what the hierarchy would control in the first place. Since all sectors are treated the same, it cannot be used to implement hierarchical access levels, and implementing a hierarchy which only affects the key hierarchy would be silly. It was therefore decided that GBDE would support a number of parallel key paths, and four was chosen as the default number which can be changed at compile time.

The fifth usability criterion was that GBDE should be able to use any byte string as passphrase, and not be limited to NUL terminated text strings. In many settings, the necessary key strength cannot be derived from a keyboard entered text string, and therefore it should be possible to use storage devices like USB-keys or smart cards to contain or contribute to the passphrase.

The final usability criterion was that GBDE should offer the best possible protection also for the user. This might require some explanation:

4.1. Protecting the user

The weakest link in any cryptographic deployment is almost always the users and their handling of key material.

As an example of this, most banks use two-man procedures to protect their vaults. Not because this prevents their employees from embezzling money — there are plenty of ways they can do that — but because it protects the employee from being a weak link which could be broken by means of physical violence, blackmail or hostage taking.

These kinds of situations mandate that analysis treat the user as a component, something which have become routine by now.

Unfortunately it is still a relatively common mistake to either leave the attacker out of the analysis, or to assume a very weak or even stupid attacker.

An example of this, is the ‘‘The Steganographic File System’’ [STEGFS], which provides a facility where a hierarchy of passphrases protect data at different levels.

The argumentation more or less goes ‘‘protect a couple of unimportant files at the lowest level, and your important files at higher levels. When captured, give them the lowest level key and deny that any more keys exist.’’

If we include the attacker in the analysis, she will soon know that the facility used is STEGFS, and consequently that multiple levels of keys are not only a possibility but to be expected: Why else would people use STEGFS in the first place ?

A user caught with a STEGFS encrypted set of data, will therefore likely be subject to pressure to release keys until the attacker is satisfied that there are no more keys. If the attacker is the police, this can now land the user in jail for up to five years in certain countries.

But even worse: if the attacker has her own ideas of what is to be found in the protected data, for instance information on weapons of mass destruction, but no such information is on the protected set, there is no way the user can ‘‘get off the hook’’ and prove his innocence: STEGFS provides no way of proving the nonexistence of any further keys.

If one studies the evidence of a real-world scenario: the plight of the junior CIA operative resident and later hostage at the Tehran embassy [IRAN], a couple of interesting insights emerge.

Very often, the user will have a tiny window in time during which it may be possible to manually activate data destruction mechanisms, or alternatively, be able to out-wait a specific timeout after which automatic mechanisms will have destroyed the data if they are not rearmed correctly.

For such a feature to offer the user effective protection it must provide a tangible feedback to the attacker that the user has destroyed the data, and can not bring it back. Effective feedback has been smoking piles of ash, but not, as related in the narrative from Tehran, piles of shredded documents.

The final GBDE design criterion was therefore that GBDE should be able to rapidly destroy key material in such a way that it can be proven that there is no hope to recover the data short of very expensive brute force methods.

5. What GBDE does not do

With any cryptographic facility, it is as important to know what it does as what it does not, this section will describe a couple of major issues which GBDE leaves to the user to solve.

5.1. Key management

It has been said that there is only one really hard problem in cryptography: the problem of key management. GBDE does not try to address that.

In all of the following we will assume that the passphrases and other key material have been handled in a sensible manner, fitting the importance of the data they are used to protect. If you put a yellow sticky note with the passphrase on a disk, no amount of cryptographic code will offer any protection.

5.2. No protection for ‘‘hot’’ disks

GBDE is only designed to protect data on a ‘‘cold disk’’. A cold disk is defined as a disk disassociated from the keys which provide access to the disk. [3]

If the GBDE protected device has been ‘‘attached’’ on a running system, parts of the key material will be present in memory and therefore GBDE’s cryptographic protection is no stronger than the protection the operating system gives to those data.

If the protected device is not ‘‘attached’’, there is no key-material present in RAM and the disk is fully protected by GBDE.

A disk on a shelf or in a courier bag is obviously also a cold disk.

It is important to note that disks in laptop computers which are only ‘‘suspended’’ can not be be considered cold if they are attached in the kernel, because the key material is present in RAM or possibly in the ‘‘suspend to disk’’ partition on the disk. [4]

While most computers offer facilities for password protection when leaving suspend mode, it should be noted that modern hardware facilities may contain wide open back doors around that protection.

For instance, FireWire/IEEE1394 capable devices are mandated to have have built in ‘‘OHCI’’ facilities which allow direct access to the entire memory space without the CPU being involved or aware of this. We

[3] Under carefully controlled circumstances GBDE can also be used as a component to build protection for ‘‘hot disks’’; that is discusse later.

[4] Interestingly, if GBDE was implemented in the disk drive and the BIOS handled the pass-phrase entry, the suspend-to-disk partition would also be protected.

estimate that using this method it would be possible to undetectably grab a snapshot of all RAM on a typical notebook computer in a few minutes. A great tool for debugging, a nightmare tool for security.

6. Cryptographic design

The usability requirements produces the overall layout of the GBDE cryptographic design which the cryptographic design must respect: A passphrase gives access to a copy of the master-key material, which again gives access to the protected sector contents.

From a cryptographic point of view, the requirements are very simple: make it as strong as possible using as few resources as possible.

While we currently have great faith in algorithms like the Rijndael/AES, and it is generally accepted that 128bit symmetric keys offer practically impenetrable security, the history of the cryptography has shown that significant weaknesses take long time to uncover.

Since the aim is that GBDE be able to protect a cold disk for a decade or more, it would be unwise to rely on our current understanding of the subject and the algorithms to remain unchallenged for the lifetime of the data.

A number of defensive measures have therefore been been included in the design criteria, in order to put sufficient margin into the cryptographic strength of GBDE.

The first criterion is that plaintext sector data should be encrypted with one-time-use (pseudo-)random keys. This measure, while relatively expensive, effectively stops differential plain-text attacks.

This forces yet a design decision since the PRN sector-keys obviously must be encrypted and stored on the disk with the encrypted plain-text. The second criterion is that these sector-keys are encrypted with a per sector ‘‘key-key’’ derived from only a fraction of a very large master key.

By only allowing a fraction of the master-key into the calculation, a penetration which manages to find the key-key for a given sector will not expose the master key, and since the sector key which the key-key encrypts is (pseudo-)random, no differential leverage exists at this level either.

The third cryptographic criterion was that a remapping of sectors should happen, so that the location of a particular plain-text sector on the encrypted volume is not predictable. Many file systems have well defined and easily predictable ‘‘super blocks’’ and allocation bitmaps which have well defined sector addresses. By mapping the locations on the disk, these sectors offer no leverage for attacks, until their encrypted location has been found by some other means.

The final cryptographic criterion was to use only standard cryptographic algorithms in good standing: Rijndael/AES as block cipher, RSA2/512 as strong one way hash and MD5 as ‘‘bit blender’’.

7. The GBDE algorithm

As can be seen from the above, the *a priori* requirements have closely circumscribed the solution space for GBDE, and the algorithms four steps follows more or less directly from the design.

7.1. From passphrase to master key

It is generally accepted that a passphrase consisting of one or more sentences in a human language only contain about one bit of effective entropy per character, and obviously they are variable length. Therefore the passphrase is passed through the SHA2/512 one-way hash algorithm. This transforms the variable length passphrase to 512 bits which contain as much as possible of the entropy in the passphrase.

These 512 bits, called ‘‘the key material’’, are used to locate and decrypt the so called ‘‘lock sector’’ which contains the master-key and various parameters.

A compile time constant, sets the number of lock sector copies supported, the default number is four. These copies are located in four sectors chosen randomly at the time when the device is initialised for GBDE usage.

The first 128 bits of the key material are used to decrypt the sector offset of the encrypted and encoded lock sector. The encrypted version of this offset can either be stored in the first sector of the device or outside the device in a file.

Once read from disk, the next 256 bits of the key material is used to decrypt the encoded lock sector using AES/CBC/256.

The final 128 bits of the key material define the order in which the fields of the lock sector are encoded. Numeric fields are encoded in little-endian byte order so that the on-disk format is portable between different architectures. Once decoded, one of the fields offer a MD5 hash checksum which can be used to prove that this is actually a valid lock sector.

7.2. Sector mapping

It follows from the above that a number of transformations are necessary on the sector location, in

addition to the cryptographic transformation of the sector contents.

The first transformation, splits the plain-text sectors into a number of zones. The size of a zone is chosen so that one sector exactly contains the encrypted sector keys for all the payload sectors in the zone. With a sector size of 512 bytes and a sector key size of 128 bits, the results in a zone size of

$$\frac{512 bytes/sector * 8 bits/byte}{128 bits} = 32 sectors$$

The extra sector with the encrypted sector keys is appended after the last of the 32 data sectors, making the zone 33 sectors long in the encrypted image.

It should be noted that the sector size for the encrypted device and consequently the zone size is a parameter set at device initialisation time.

The second transformation is a rotation by an integral number of sectors. The number is randomly chosen at device initialisation time and recorded in a field in the lock sector. This step addresses the design requirement that the location of the encrypted sector should not be trivial to determine.

The third transformation inserts the four lock sectors into their locations. The locations of these are chosen at random at device initialisation time, and stored in fields in the lock sector. It follows quite obviously from this, that each lock sector must contain information about the location of all lock sectors, in order to map around them.

Finally an offset is added which determines where the encrypted image starts in the underlying device.

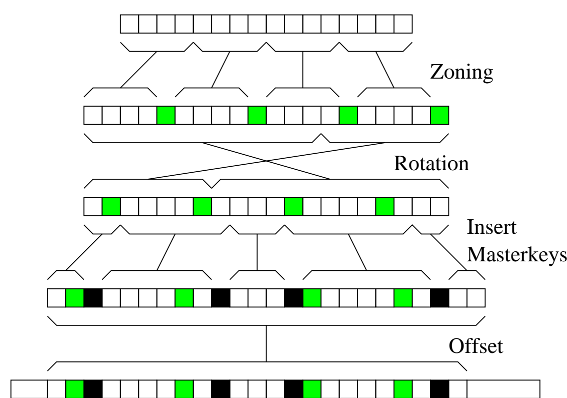

fig 1: mapping operations.

The offset mapping serves two purposes, it allows the first sector of the underlying device to be reserved to contain the encrypted locations of the lock sectors, but it also allows the administrator to use only a part of the underlying device for GBDE encrypted data.

7.3. Plausible denial

This could be used to implement ‘‘plausible denial’’ by embedding the GBDE partition inside some data which credibly can be claimed to be something else. Since high entropy data are rare in the wild, special care is needed to build a convincing cover story to explain the existence of otherwise unexplained random looking bits.

Most computers today come pre taxed with a Microsoft operating system, and this could be used for the cover story: Flush all free space in the Windows partition with random bits, locate a long stretch of free space in the partition and use that to contain the GBDE encrypted data. If no other evidence betrays the existence of the GBDE partition, it should be possible to deny its existence.

A truly paranoid setup would leave the computer configured to boot the Windows system by default, and locate the GBDE data in such a way that it would be destroyed by the act of doing so.

7.4. ‘‘Key-key’’ derivation

For each sector to be processed, we need to derive the key-key which the sector key is encrypted with. For this purpose the lock sector contains two fields, the ‘‘salt’’ which is a 128 bit (pseudo-)random number, and the master key which is a 2048 bit (pseudo-)random number. Both of these are generated when the device is initialised for GBDE usage and not subsequently changed.

In order to make the format architecture invariant, the the sector address is encoded as a little-endian 64 bit integer

$$sect = LE64(sectoraddress)$$

and then, surrounded by the salt, run through the MD5 algorithm which acts as a ‘‘bit-blender’’. The salt is necessary to prevent pre-computation of a dictionary of all possibly sector addresses as an attack vector.

$$index[16] = MD5(salt[0...7] + sect + salt[8...15])$$

The resulting 16 bytes of index are used to ‘‘cherry-pick’’ 16 bytes from the master key, which are run through MD5 with the sector address inserted in the middle: [5]

[5] In both cases the encoded sector address were put in the middle of the MD5 input data only for reasons of symmetry, cryptographically it makes no difference.

$$keykey = MD5(\; masterkey[index[0]] + \ldots + masterkey[index[7]] + sect + masterkey[index[8]] + \ldots + masterkey[index[15]])$$

Some research have cast doubt about the cryptographic qualities of the MD5 algorithm [RSAMD5]. The questionable property, how hard it is to generate collisions, is of no concern here: MD5 is merely used as a "bit blender" and unlike when MD5 is used for authentication purposes, there is no value to the attacker in producing a collision with a different input.

7.5. Sector data encryption

To encrypt and write a sector of plain-text data, the key-key is derived as above, and the sector containing the encrypted sector keys is read into memory.

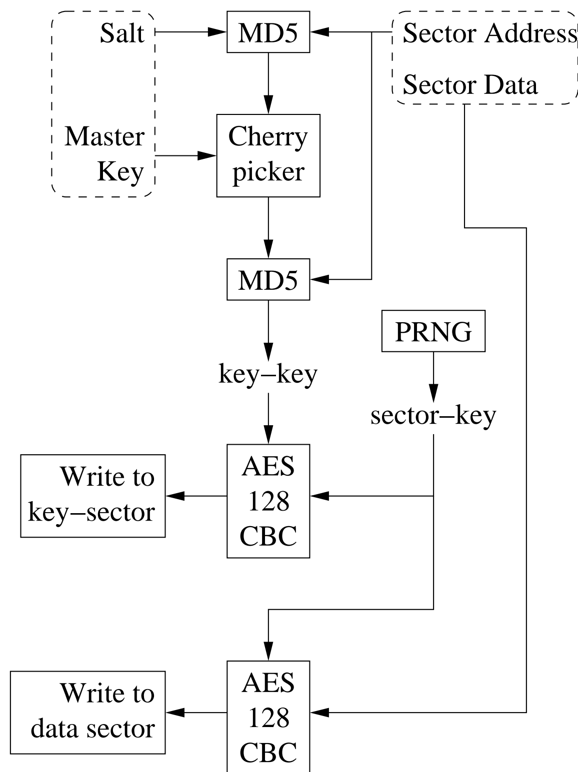

fig 2: write operation

A new sector key is generated with the standard (P)RNG facilities in the kernel and the sector data encrypted using AES/CBC/128.

The sector key is encrypted with AES/CBC/128 using the key-key and inserted at the correct place in the key sector.

Both the key sector and the encrypted data sector are written to disk after which the temporary storage is cleared and the transaction is complete.

To read and decrypt a sector, the key sector and the encrypted sector are read into memory, the key-key generated and the sector key decrypted with it. The sector key is used to decrypt the encrypted sector data, the temporary buffers erased and the transaction completed.

Since the sector data is encrypted with a (pseudo-)random key, there is no need to use any particular initialisation vector for the encryption process and therefore a constant IV of zero bits is used [6].

7.6. Performance optimisations

If implemented strictly according to the above description, GBDE would take a drastic performance hit because all I/O requests, no matter their size would explode into two requests per sector on the underlying device for each sector in the request. Even with the caches built into modern disk drives, this would lead to a drastic performance hit.

Instead the request is split into a number of "work packets" which are characterised by the fact that all data sectors in the work packet are consecutively laid out on the device, this also implies that they are in the same zone and therefore need the same key sector.

Furthermore to avoid re-reading a key sector which has just been written to disk, a small cache is used to keep copies of the encrypted key sectors which have recently been used. The size of the cache grows linearly with usage of key sectors, and decays with 10% every second [7]

Furthermore, the logical sector size of the encrypted device can be set to a power of two multiple of the underlying devices sector size. Using larger sectors means fewer keys, larger zones and consequently larger work packets.

For paging spaces, the VM page size should be used as logical sector size, for file systems the smallest allocation unit should be used. For UFS/FFS this is the so called fragment size, and it is typically 2048 bytes.

In retrospect zero bits were a bad choice since any sector on the encrypted device which contains all zero bits show up as a distinct signature on the encrypted side. This does not introduce a vulnerability inside the design envelope, but it will nonetheless be addressed in the next revision of GBDE since a fix will not affect the installed user base.

[7] It is not possible to use the regular kernel buffer cache because GBDE operates below the SPECFS layer. Furthermore the buffer cache offers no means to clear the buffers contents before it is recycled.

8. Administrative operations

In order to use a device with GBDE it must be initialised first. The initialisation will generate the contents of one lock sector and encode, encrypt and write it to the device.

While this is technically enough to get started, it is highly recommended that the entire device be filled with (pseudo-)random bits before writing the initial lock sector, since this will prevent an attacker from using the previous possibly less random sector contents to eliminate a large fraction of the sectors from an attack.

Unfortunately, it takes considerable amount of time to generate (pseudo-)random bits for modern sized disk devices, and this process can take hours or even days to complete.

To use a GBDE device, it needs to be ‘‘attached’’. This involves presenting the passphrase, and using that locate and decrypt, decode and validate the lock-sector.

The opposite process, detaching the GBDE device is mainly a question of erasing all traces of the lock sector and key sector copies from memory.

Since the location of all lock sectors is recorded in each lock sector, it is possible for any user with a valid passphrase to overwrite any lock sector.

One operation is writing a new lock sector protected by a new passphrase. If done for the lock sector which the user has a passphrase for, this amounts to changing the users passphrase.

Another option is to overwrite the lock sector(s) with zero bits. This disables the passphrase(s), a condition which GBDE will report specifically on, in order to provide the tangible feedback of destroyed data discussed earlier.

9. Attacking GBDE devices

Given a media containing data protected with GBDE, there are two avenues for attacking the algorithm: From the top, trying to get hold of the masterkey and salt from one of the lock sectors, from the bottom, trying to derive them from decrypted sectors, or by trying to guess the passphrase.

9.1. Top down attack

The first challenge is to identify which sectors contain what.

There is currently no known algorithms or statistical methods which are able to tell bits produced with AES/CBC/256 and AES/CBC/128 apart, nor is it possible to tell them apart from random bits, so the only way to locate the lock sectors is by doing a brute force search.

If the encrypted location of the lock sectors is available, it may be faster to brute-force the 128 bits which encrypt the location than to search all possible byte offsets on the disk.

Attacking the lock sectors means finding the 256 bit encryption key for the encoded data, and the permutation of the 12 fields in the lock sector.

Given that one of the fields is a MD5 hash which will can be used to test for a hit, the worst case work to brute force the lock sector is therefore somewhere in the neighbourhood of:

$$W_{AES/CBC/128} \cdot 2^{128} + (W_{AES/CBC/256} + W_{MD5}) \cdot 2^{256}$$

or

$$N_{bytes_on_device} \cdot (W_{AES/CBC/256} + W_{MD5}) \cdot 2^{256}$$

or, if the attacker by some other means have located the lock sector:

$$(W_{AES/CBC/256} + W_{MD5}) \cdot 2^{256}$$

All of which are practically infinity by todays standards.

9.2. Bottom up attack

Attacking from the other end, so to speak, we will assume that the attacker either knows the plain text of a number of sectors or is able to recognise it by some algorithm, and let us give him the benefit of knowing all the parameters which goes into the mapping of sectors.

Doing a brute force on one of the sectors yields the sector key which can be used to brute-force the encrypted copy of the sector key, which again yields the key-key for this particular sector.

Having obtained the key-key for one specific sector, the next step is to find the 16 unknown bytes of the 24 bytes input sequence to MD5 which generate this key-key as output. (The middle 8 bytes is the little-endian encoded sector number, which we assume the attacker already knows).

In the worst case, the workload so far is:

$$2 \cdot W_{AES/CBC/128} \cdot 2^{128} + W_{MD5} \cdot 2^{128}$$

The attacker now has the value of up to 16 bytes (there may be duplicates) of the 256 bytes in the master key, but he does not know where those 16 values are located in it.

If the attacker brute-forces multiple sectors, he can try to brute-force the 128 bit salt using the known master

key bytes as truth detector. The exact number of sectors he needs to brute force is very hard to predict but let us assume the worst case value of two. The worst case work necessary to do this is:

$$W_{MD5} \cdot 2^{128}$$

Knowing the salt the attacker can now predict which bytes in the master-key are involved in the generation of the key-key for each sector on the disk, and may be able to decrypt a number of sectors based on the subset of the master key he knows so far.

To extend his knowledge of the master key he will have to attack more sectors using brute force, but the search space is reduced by the already known bytes of the master-key so each new sector may be obtainable along with a byte of master key by a worst case effort as low as:

$$2 \cdot W_{AES/CBC/128} \cdot 2^8 + 2 \cdot W_{MD5}$$

at which point the door is wide open.

Summing up, the total lowest worst case effort, given known plaintext is in the neighbourhood of

$$2^{129} \cdot W_{AES/CBC/128} + 2^{128} \cdot W_{MD5}$$

which is also practically infinity by todays standards.

9.3. Attacking the pass-phrase

As already mentioned, using human language text for pass-phrases yields only about one bit of entropy per word, so a well written 64 bit entropy pass-phrase could be:

> Blow, winds, and crack your cheeks! rage! blow!
> You cataracts and hurricanoes, spout
> Till you have drench'd our steeples, drown'd the cocks!
> You sulphurous and thought-executing fires,
> Vaunt-couriers to oak-cleaving thunderbolts,
> Singe my white head! And thou, all-shaking thunder,
> Smite flat the thick rotundity o' the world!
> Crack nature's moulds, and germens spill at once,
> That make ingrateful man!
> [LEAR]

Given that few actors can deliver that correctly over the lime-lights, we can expect that the average pass-phrases will offer less entropy, and dictionary attacks are consequently not only feasible, but to be expected.

The worst case work required to test one passphrase candidate amounts to:

$$W_{SHA2/512} + W_{AES/128/CBC} + W_{disk-read} + W_{AES/256/CBC} + W_{MD5}$$

while the best case work to reject it (because the decrypted location of the lock-sector would be found to be outside the physical media) is only:

$$W_{SHA2/512} + W_{AES/128/CBC}$$

The LOOP-AES [LOOPAES] facility in Linux offers an ''iteration count'' which adds N thousand iterations of AES/256 to the pass-phrase preprocessing path in order to make such an attack more expensive.

With the increasing speeds of hardware and the availability of cryptographic co-processors, protecting a weak pass-phrase for a decade or more in this fashion is neither realistic nor prudent, and GBDE therefore does not implement a similar option.

A better way to frustrate a dictionary attack is to combine the pass-phrase with a high-entropy token,[8] for instance 1024 bits generated in a suitable random way. This token could be stored on a authentication device or removable storage device such as an USB-key.

10. Known weaknesses

No armour is impenetrable, and no dragon without a soft spot. Three issues have been identified where GBDE has less than perfect performance.

10.1. The static master key

The fact that the lock-sector contents does not change with a change of passphrase means that once a person has once had access to a device, it is not possible to reliably revoke that access by changing the pass-phrase: He would be able to restore a copy of the lock-sector for which he knows the pass-phrase, and thus gain access to the device. All things being equal, this is only marginally more problematic than the fact that the user may also have kept copies of the plain-text data. If this is a real concern, the proper mitigation is to copy the data off the device, reinitialise the GBDE lock sectors, and copy the data back. In the future an ''fast re-encrypt'' operation would help make this a less painful procedure.

10.2. Huge appetite for random bits

Because sector keys are single use (pseudo-)random bits, GBDE can consume up to sixteen PRNG bytes for every 512 byte sector written to the disk, or 32 kilobytes per megabyte of plaintext. It is obviously important that these PRNG bits are high quality and that the source cannot be manipulated or tampered with, in particular when the device is first initialised where the master-key and salt bit-strings are chosen.

---

[8] This principle is often referred to as ''something you have + something you know''

If the attacker can predict or manipulate the bits assumed by GBDE to have random properties, attacking the protected data could become trivial.

It follows from this that a PRNG which is not regularly perturbed by outside entropy sources would be a very bad source for GBDE, because it would be possible to predict all future bits once the state of the generator was determined once.

In FreeBSD the kernel random facility is based on the industry standard Yarrow algorithm which uses cryptographic functions to churn out high entropy bits based on entropy collected from hardware sources [YARROW].

10.3. The cleaning lady copy attack

An attacker in position to make a bit-wise copy a GBDE protected media on a regular basis would be able to gain a head-start on attacking the device by being able to monitor which sectors change and which do not from one snapshot to the next.

Given just a few copies spaced a day apart, an attacker could likely pinpoint the location of the UFS/FFS super blocks because of their strict geometric distances, and the lock sectors may stand out after a longer time period. In the analysis of attacks on cold media, we more or less assumed that the attacker had this head-start, but the full impact of this kind of attack can only be judged for a specific file system or content type.

A log structured file system would probably be harder to unravel this way because data and metadata are all intermingled in sequential write operations.

11. GBDE as a component

A facility like GBDE is obviously only a component in any real security implementation. The main interface to GBDE as a component is the passphrase and the administrative operations.

11.1. Typical laptop deployment

In an organisation, one possible way to deploy GBDE on laptop computers could be the following:

The laptop is installed with operating system and disk space is set aside for the GBDE partition(s). The security department initialises the GBDE partition with a top-secret company passphrase. The second lock sector is initialised using a passphrase which the branch-office manager has access to, and the third lock sector is initialise with a new passphrase which the user is given access to.

To further improve security, the security department could copy the first two encrypted lock sectors to a floppy disk and replace them with random bits.

Should the user loose his passphrase or if he decides to activate the lock sector destruction function in some situation, the floppy disk can be used to restore the lock sectors and the contents of the device can be recovered from there.

Needless to say, destroying the lock sectors adds little security if the adversary has access to a backup copy, so while the floppy disk does not need by definition need to be stored in a safe, it should obviously not be stored next to the encrypted device either.

11.2. Server deployment

For server use the issue is slightly more complex. It is usually desirable to be able to keep the passphrase stored on the server so that it will be able to boot automatically, but at the same time make it impossible for an attacker to get hold of the passphrase and the protected disk at the same time.

One way to implement this, is by using a ‘‘weak link/strong link’’ method similar to that use in ‘‘permissive action links’’ in atomic weapons [PAL]:

The server is installed in an small enclosure with a very sensitive and very fast intrusion detection. If an intrusion is detected (breach of the weak link), the computer will destroy the passphrase (disabling the strong link), thereby protecting the encrypted data.

Needless to say, there needs to be an internal power source which can provide sufficient power to reliably destroy the passphrase, in case the attacker starts out by cutting the power supply to the enclosure.

12. Performance impact

With an emphasis on strong crypto, performance will always suffer, and GBDE is no exception.

12.1. Throughput

The most expensive part of the GBDE design is the stored encrypted sector key, which worst case forces two disk operations per sector operation.

The first set of measurements try to gauge how much the logical sector size affects the performance. To eliminate as much of the physical aspect, a test of 100 megabytes sequential read and write using 1 megabyte operations were run with varying GBDE logical sector sizes and without GBDE.

The hardware used was an AMD Athlon 700MHz CPU using an IBM DTLA-307015 disk. For workstation or server use, this is somewhat behind the state of the art, but its performance is pretty close to that of a modern laptop computer.

Operations Mode	seq. read kB/s	seq. read stddev	seq. write kB/s	seq. write stddev
unencrypted	36141	28	27915	738
512 bytes	7326	12	3447	30
1024 bytes	8088	41	6023	50
2048 bytes	7880	32	5082	25
4096 bytes	8140	176	6061	51
8192 bytes	8849	37	6597	8

As can be see from the numbers, GBDE runs at approx 20-25% of the unencrypted speed if the logical sector size is one kilobyte or above.

Why using 2048 byte logical sectors is slower than both 1024 and 4096 is somewhat of a mystery. We speculate that it may be related to memory management arena fragmentation, or possibly a cache interaction with the UMA memory allocator.

12.2. Latency

Given that the encryption operations are very fast compared to the seek times of contemporary disk hardware, the expected behaviour with respect to I/O latency is that operations with a high degree of locality will take twice as long, and for operations where the disk mechanism incurs arm settling time, the increase in I/O latency will be lost in the noise.

An attempt was made to measure GBDE's impact on I/O latency using a modified version of the `diskinfo -t` tests.

The numbers collected confirmed that highly local operations roughly double their I/O latency, typically from 150µs to 300µs. But for less local operations the measurements were so noisy that statistically they prove nothing useful, one way or the other. A ''hairy-eyeball'' judgement of the numbers found nothing which would disprove our hypothesis and found them generally compatible with it.

We believe that the noise was due to the complexity of the situation, where GBDE's sector cache, the disk drivers use of the disksort() function and the disk drives internal optimisations result in very significant changes in the order of things for even very minor changes to the driving program.

12.3. Other indications

No attempt to quantify CPU usage more precisely than ''will eat a lot of CPU cycles'' have been attempted, the expectation is that this will improve when hardware assisted cryptographic functions become available.

The author's personal home directory has been GBDE encrypted for more than 6 months at this point in time, and he has neither found the performance annoyingly slow nor lost any files in this period.

13. Future improvements

The current implementation of GBDE does not take advantage of hardware assisted cryptographic processing. Once AES capable hardware accelerators become available the GBDE workflow engine should be changed to take advantage of them.

A number of additional administrative operations can be imagined, for instance a mode where a device is initialised and attached in one operation which does not write the lock sectors to the media thereby preventing later re-attachment. Such a mode would be useful for devices used for paging or temporary file systems. A command line option to save and restore lock sectors to a file would be convenient too.

Fast re-encryption operation: it is possible to write code to change the master key and salt which only decrypt and re-encrypt the sector-keys. This would be two to three orders of magnitude less work than re-encrypting the entire volume.

14. Availability

GBDE is distributed under the ''two-clause BSD license'' and will be maintained in the publicly available FreeBSD CVS repository, from where we encourage other operating systems to adopt it.

15. Conclusion

GBDE was designed and implemented as a cryptographic facility which operates on the raw disk level with real-world compatible semantics.

The operational experience so far is good, and initial user feedback has been very positive.

The reviews have so far agreed that the cryptographic design is sound and very strong approching overkill for most ''normal'' applications.

It is impossible to set a guaranteed protection time on any cryptographic algorithm, but we belive that if AES/128 retains at least 80 bits of effective strength,

GBDE will protect its payload against any terrestial attacker for at least 10 years and probably also 25 years

. . . provided the attacker can not guess that the passphrase was: “Det er svært at spå, især om fremtiden.” [9]

16. Acknowledgements

The author would like to thank:

Robert Watson for organising funding and subsequently translating danglish to the proper red-tape protocol for the involved paper tigers.

Lucky Green for, in addition to being incredibly patient and helpful, trusting his data to GBDE from a very early stage.

David Wagner for his insight, review and in particular for reiterating the obvious until it finally registered.

Bruce D. Evans for his enthusiastic resistance and competent technical hindrance. It would have been much to easy to cut corners if it were not for people like Bruce, much appreciated!

Also thanks to all the people who have listened attentively while I have bored them with incoherent explanations and fuzzy drawings, and people who have helped stamp out “Danglish” from this paper, *Flemming Jacobsen* and *Gregory Sutter* in particular.

And finally a big thanks to the *FreeBSD crew* for putting up with me and my crazy ideas.

17. References

[CGD]
The “cgd” facility in NetBSD: http://netbsd.gw.com/cgi-bin/man-cgi?cgd+4+NetBSD-current

[CPRM]
CPRM was an attempt to mandate Digital Restriction Management features be implemented in all read-write storage devices, such as ATA hard disks.

For a full time line of its emergence and defeat see this summary article on “The Register” 18-02-2001: “CPRM on ATA - Full Coverage” By Andrew Orlowski (http://www.theregister.co.uk):

[DOJ0227]
“The Federal Bureau of Investigation’s Control Over Weapons and Laptop Computers”

Report No. 02-27, August 2002, Office of the Inspector General reports 317 laptops lost over 28 months. This is about 2% of the FBI’s inventory, and a rate of one every three days.

[IRAN]
“Held Hostage in Iran - A First Tour Like No Other” William J. Daugherty, 1996 http://www.cia.gov/csi/studies/spring98/iran.html

[LEAR]
Shakespeare: King Lear, Act III, scene 2.

[LOOPAES]
The “loop AES” facility in Linux: http://loop-aes.sourceforge.net/loop-AES.README

[PAL] “Permissive Action Links” Steven M. Bellovin http://www.research.att.com/~smb/nsam-160/pal.html

[PGP]
“Pretty Good Privacy”, originally written by Phil Zimmerman, subsequently given the dot-com runaround. A good place to start: http://www.pgpi.com

[PRIV]
Private communication. This policy is confidential, so the source cannot be revealed.

[RSAMD5]
RSA Laboratories’ bulletin Number 4 - November 12, 1996 “Recent Results for MD2, MD4, and MD5” ftp://ftp.rsasecurity.com/pub/pdfs/bulletn4.pdf

[STEGFS]
“The Steganographic File System” Paper by Ross Anderson, Roger Needham and Adi Shamir. http://www.mcdonald.org.uk/StegFS/

[THEREG]
See for instance, these articles on “The Register” 24-03-2000: “Sneak thief steals state secrets in MI5 laptop” 06-04-2000: “Third secret-packed official notebook nicked” 18-04-2000: “FBI admits loss of ’top secret’ laptop” and on a slightly different theme: 23-08-2001: “$15k reward offered for lost laptop” (http://www.theregister.co.uk):

[YARROW]
Yarrow is a secure pseudorandom number generator designed by Bruce Schneier and John Kelsey. http://www.counterpane.com/yarrow.html

[9] “It is difficult to predict, in particular the future”
—Robert Storm Petersen (1882 - 1949)