TIMO SIVONEN

# measuring performance of FreeBSD disk encryption

Timo Sivonen is a Senior Consultant at International Network Services (INS). He lives in the U.K. with his wife and son.

*timo.sivonen@ins.com*

**DISK ENCRYPTION IS ONE OF THOSE** services absolutely invaluable to laptop owners and possibly even suspicious if used by others. Yet, as with many other security services, you have to ask yourself what is the threat you are trying to protect against or the problem you are trying to solve.

Low-level disk encryption, which encrypts everything on the raw partition including the file system, does keep your files secure if the passphrase to open the partition is not known. Yet, if the encryption layer lies below the file system, almost the only way to create an encrypted backup of an encrypted file system is to copy the partition to an image file with dd(1), which is both cumbersome and inefficient.

In my day job I have to store confidential client data on my laptop and I have to back the laptop up regularly. However, data encrypted on the hard disk must remain encrypted on backups. Furthermore, I prefer multiple 650 MB encrypted file systems to one 4 GB volume, since I can back the smaller volumes up on individual CDs as opposed to using more cumbersome tapes. Like everyone else, I also use a flash drive, but instead of carrying around several flash drives for different operating systems, I wanted to consolidate my flash drives to one and copy files between UNIX boxes using an encrypted file system without losing UNIX/Windows interoperability offered by FAT32.

Since version 5, FreeBSD has featured GEOM-Based Disk Encryption (GBDE). In brief, GBDE is a software-only disk-encryption service that uses AES-128 to encrypt the contents of the designated raw device and the master encryption key is stored under AES-256. GBDE seemed to offer exactly what I was looking for except for the fact that all documentation pointed toward encrypting raw partitions. However, FreeBSD also has a facility called md(4), or memory disk, which, among other things, allows you to read and write image files as raw devices. Combined with newfs, one is able to write a file system on the image and mount it like any other disk device. All that is required is to set up the GBDE layer on top of md, write a file system on the GBDE device, and mount the device.

## Setting It Up

The initialization of a new encrypted memory disk is a relatively straightforward operation, although a few caveats do exist. (Allocate the image file, create the memory disk, label the device, initialize the encrypted device, attach the encrypted device, create the file system on the encrypted device, and mount it.) The most critical advice is to *not* enable UFS soft updates, as these may cause devfs to lock the (GBDE-encrypted) file system. In other words, a locked file system is permanently busy and can't be unmounted even in system shutdown, which ultimately may corrupt the file system. This may not happen if running FreeBSD normally, on bare metal hardware, but it is certainly possible under VMware. Later in this text we will discover that not using soft updates has little impact on file system performance.

A GBDE-encrypted file system is created on a memory disk in seven steps:

```
1.  # dd if=/dev/random of=/home/user/gbdeimg bs=1m count=650
2.  # mdconfig -a –t vnode –f /home/user/gbdeimg
    md1
3.  # bsdlabel –w md1 auto
4.  # gbde init /dev/md1c –L /etc/gbde/gbdeimg.key
                Enter new passphrase:
    Reenter new passphrase:
5.  # gbde attach /dev/md1c –l /etc/gbde/gbdeimg.key
    Enter passphrase:
6.  # newfs /dev/md1c.bde
7.  # mount /dev/md1c.bde /mnt
```

The explanation of each step is as follows:

1. Allocate the disk space by writing 650 MB of random data from /dev/random to the designated image file.
2. Create the memory disk, through which the disk image will be accessed. Note that mdconfig(8) will print out the assigned md device, unless explicitly told which device to use.
3. Label the newly created memory disk to enable creation of the encryption layer and the file system at a later stage.
4. Initialize the encryption layer. Since the encryption key is specified separately from the disk image, those really concerned about their laptop security can save the encryption key(s) on a separate flash drive, which must be attached and mounted in order to prepare and mount encrypted partitions.
5. Once the encrypted device has been initialized, it has to be attached in order to write the file system on the encrypted device. This operation will also create a new instance of the disk device with the .bde suffix to denote GBDE encryption. Hence, all file system operations must be made with the .bde device.
6. Write the file system on the encrypting device (i.e., md1c.bde in the example that follows). Remember to not enable soft updates, since devfs may become upset by this. Furthermore, it was discovered that there may not be any significant performance improvement over UFS file systems that do not use soft updates.
7. Once a file system has been created, the encrypting device can be mounted normally. All updates to the file system are written to the image file, which can be backed up using tar or cpio or burned on a CD when unmounted and taken off-line.

One unhappy discovery in this journey was that an encrypted file system cannot be attached from read-only media. You can create a memory disk from a read-only image, and you can mount UFS file systems read-only, but you cannot attach an encrypted file system if its disk image resides on, for example, a CD-ROM. This discovery was a slight setback but not critical: After all, PGP disks formatted with NTFS must be copied from the backup media to a disk first, since NTFS cannot be mounted read-only either.

## Choices and Performance

FreeBSD 6 introduced a new encrypted file system, GELI (GEOM_ELI cryptographic GEOM class). Unlike GBDE, which is a software-only facility, GELI utilizes the crypto(4) framework and is able to use encryption hardware if available.

GELI also gives more choice in algorithm selection and key length. Whereas GBDE only uses AES-128 for encrypting the disk contents, the users of GELI can choose from 3DES, AES, and Blowfish with key lengths of 128 or 256 bits. 3DES has a fixed key length of 192 bits. With this kind of selection it may seem difficult to choose the right encryption algorithm and balance security with performance.

To answer these questions and to be able to make educated decisions on which disk encryption to use, or which algorithm and how long a key to select, I devised a test plan to give some insight into the performance of FreeBSD disk encryption. Understanding that the results would be affected by the type of hardware available, even when executing tests on an otherwise idle system, one has to accept a certain distribution in results since, after all, UNIX is a time-sharing operating system and does not guarantee any throughput time for a command or a system call. My plan was to measure processing times for write and read operations when writing and reading a single 100 MB file or writing and reading 100 1-MB files. The files would only contain random binary data as read from /dev/random. Hence, the following tests were conceived:

- Writing and reading a 100 MB file and 100 1-MB files using GBDE and GELI on a memory disk.
- Writing a 100 MB file and 100 1-MB files using GBDE and GELI/AES-128 on a raw disk partition. The purpose of this test was to establish the baseline on what type of throughput one may expect from encrypting file systems. Without the md layer in the way, one would expect a visible improvement in performance.
- Writing a 100 MB file and 100 1-MB files using GELI on a memory disk, with a variety of ciphers. The purpose of this test was to compare the performance of GELI when a cipher other than the default AES-128 is used.

Each write and read test was repeated three times and an average was calculated to get an approximate time of how long it would take to process an operation. Since the absolute processing times depend on the underlying hardware, one should not measure absolute seconds but, instead, compare approximate processing times among different disk-encryption methods.

## The Results

The test system was an IBM ThinkPad T21 laptop with an 850 MHz clock. The boot disk was a 40 GB Fujitsu MHV2040AH and the test disk was a 6 GB IBM DARA206000 running at 4,200 rpm. The tests were conducted in

multi-user mode by writing files from the boot disk to the test disk and vice versa. There were no encrypted file systems active on the boot disk. The tests were made using FreeBSD 6.1.

The most interesting test was to write a 100 MB file of random data and 100 files of 1 MB of random data to a directory. This test was conducted for GBDE, GELI, and a plain UFS file system on a memory disk. The test results are illustrated in Figure 1.

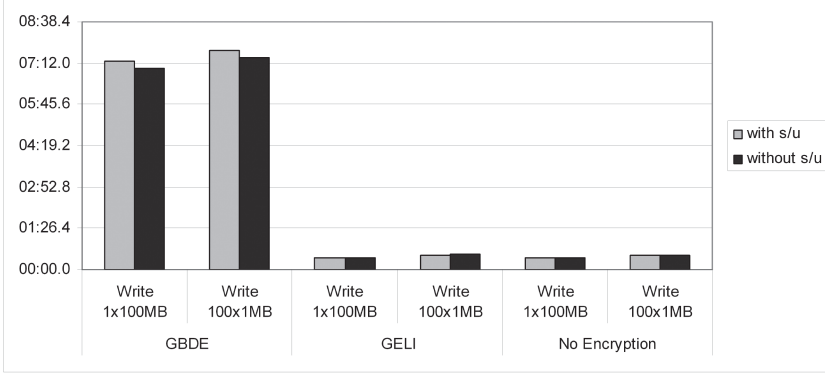


FIGURE 1: WRITING TO AN MD DEVICE

According to these results, GBDE does not perform very well at all when writing on a memory disk. In comparison, write performance of GELI is roughly equivalent to plain UFS on a memory disk. One should also note that enabling or disabling soft updates makes practically no difference to performance.

Since results on a memory disk showed a visible difference in performance between GBDE and GELI, it was measured whether there is any difference in performance for these two disk-encryption systems when writing on a raw partition, without using a memory disk layer. The results can be found in Figure 2.

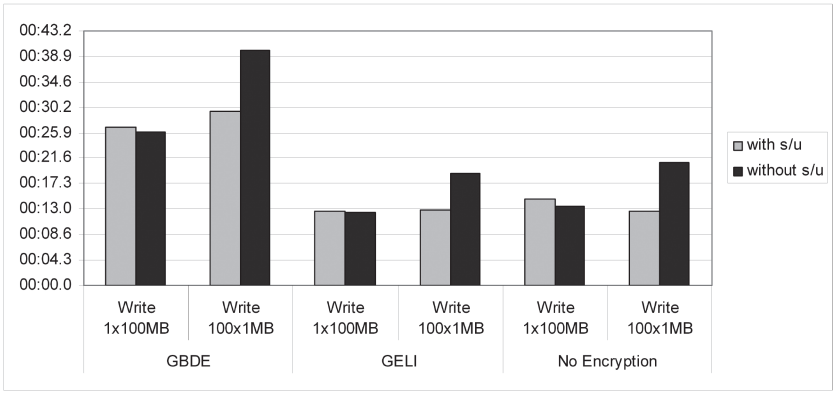


FIGURE 2: WRITING TO A DISK DEVICE

The results show that GBDE is still visibly slower than GELI but the difference between the two is no longer as dramatic. However, I am unable to explain why GBDE performs so poorly with an md device, whereas the performance difference with GELI on a raw partition is not that significant.

As one would expect, reading from a memory disk is much faster than writing (Figure 3, next page). There is little difference between GBDE and GELI, although GBDE may be slightly slower than GELI or a plain memory disk. In fact, the results show that on a moderately fast processor the time spent encrypting or decrypting is negligible compared to the disk transfer rates.

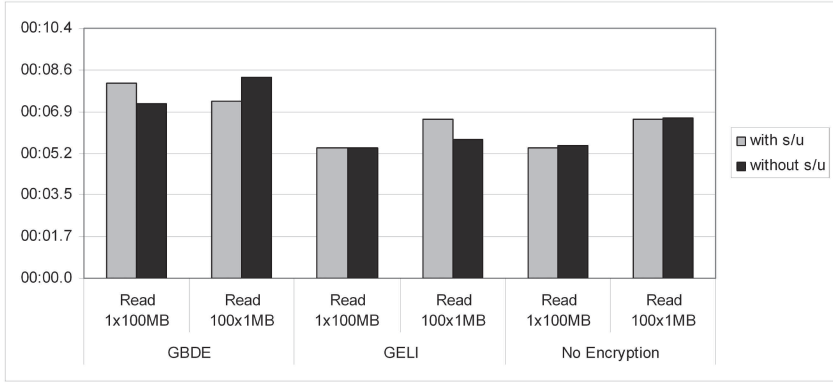


FIGURE 3: READING FROM AN MD DEVICE

The final test was to measure whether different encryption algorithms or key lengths would affect the performance of GELI in any way. The presumption was that AES and Blowfish would probably perform better than 3DES when encryption operations are performed in software. However, since no crypto hardware was available for the testing, 3DES was likely to be the worst performer.

The first presumption of performance seemed to be correct when write times using different ciphers were measured (Figure 4). 3DES-192 was clearly slower than AES-256 or AES-128 when writing a single 100 MB file or 100 1-MB files. This result was completely expected, since 3DES has to do three crypto operations (i.e., encrypt, decrypt, and encrypt again), whereas AES does only one.

A more interesting observation was the relative performance of AES-128 and AES-256. The processing times of these two were practically the same, which leads to the conclusion that one might as well use AES-256 with GELI, since the security benefits of a longer key are much bigger than the insignificant processing impact the longer key might have.

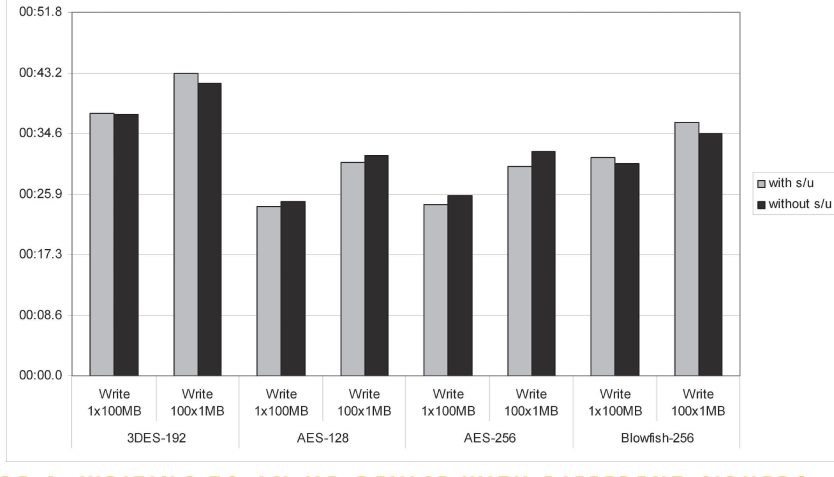


FIGURE 4: WRITING TO AN MD DEVICE WITH DIFFERENT CIPHERS

The other surprise was the performance of Blowfish 256. I would have expected Blowfish to be on a par with AES but, interestingly enough, it seems to fall between 3DES 192 and AES. Unless Blowfish has cryptographic properties that AES does not have, one would be tempted to prefer AES to Blowfish. However, a test with a larger sample may be required to determine whether AES actually is faster than Blowfish or whether the result was only a fluke.

## Conclusions

An approach using GBDE- and GELI-encrypted file systems with a memory disk, md(4), was presented. The advantage of this method is the ability to create several encrypted file systems of different sizes on a normal UFS file system. With an encrypted file system of 650 MB it is possible to back up the encrypted image on a CD and save the contents of the file system in an encrypted format. Furthermore, assuming that the UFS file system is large enough, it is possible to create new encrypted file systems and mount them on an ad-hoc basis.

The relative performance of GBDE and GELI was also discussed. It was discovered that GBDE is significantly slower than GELI using AES 128 on a memory disk. GBDE was slightly slower than GELI on a raw disk partition but there is no major performance difference between the two. This leads to the conclusion that GBDE is unsuitable for use on a memory disk and, if used, its use should be limited to removable devices such as flash drives and floppies.

Since GELI uses the crypto(4) framework and has multiple ciphers, the relative performance of different ciphers was also measured. No crypto hardware was available for these tests. It was discovered that AES with a 256-bit encryption key performed as well as AES using a 128-bit key, thus leading to the conclusion that one should be using the longer key because of its stronger security.

It was also discovered that AES performed better than Blowfish or 3DES, thus making it the cipher of choice. This observation may have resulted from the small sample, and further investigations may be called for.