Encryption at rest using eCryptfs

In chapter 8, I have discussed setting up encryption at rest using AWS ‘model C’ or a model where AWS manages keys as well as method of encryption for you. Everything is ‘point and click’. It is the most inflexible yet most convenient, easy and fast way of setting up encryption at rest. However, there are situations where you need more control over the method of encryption and also over key storage. Open source file system eCryptfs can be used effectively with Hadoop to provide encryption at rest. In this download section, I will discuss the installation and step-by-step implementation of eCryptfs for Hadoop encryption at rest.

**eCryptfs** (***E****nterprise****Crypt****ographic****F****ile****s****ystem*) is a POSIX-compliant encrypted file system that is part of the Linux Kernel. eCryptfs provides filesystem-level encryption that can be configured for an existing partition and can applied selectively on a per-file or per-directory basis. eCryptfs is included as a ‘package’ (or application) with linux-based open source operating system Ubuntu; but if you are using another open source linux implementation, then you need to download appropriate versions of packages ‘ecyptfs-utils’, ‘keyutils’ and ‘trousers’.

For my example, I have used CentOS 6.4 and had to download and install the packages as following (‘#’ is the command prompt):

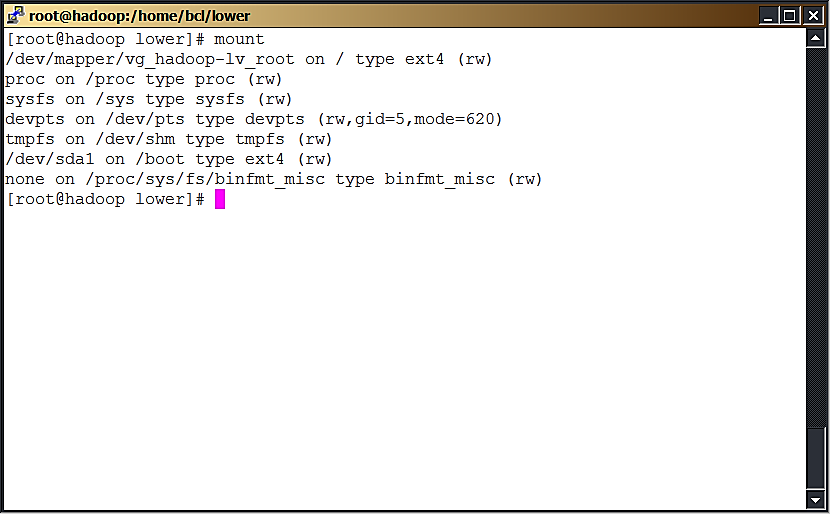
# yum install keyutils-1.4-4.el6.x86\_64.rpm

# yum install trousers-0.3.4-4.el6.x86\_64.rpm

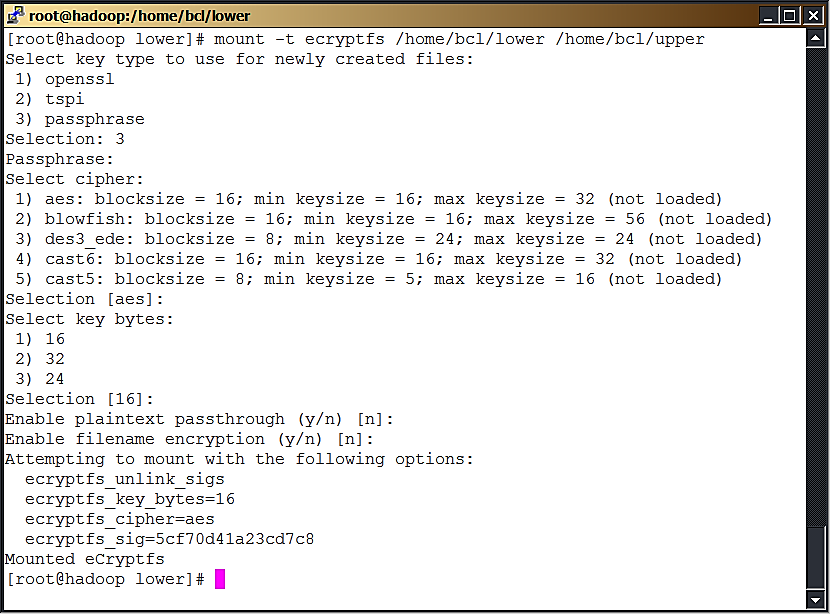
# yum install ecryptfs-utils-82-6.el6\_1.3.x86\_64.rpm

To test the install, I created 2 directories ‘lower’ (to hold unencrypted data) and ‘upper’ (to hold encrypted data). I copied 2 text files in the ‘lower’ directory.

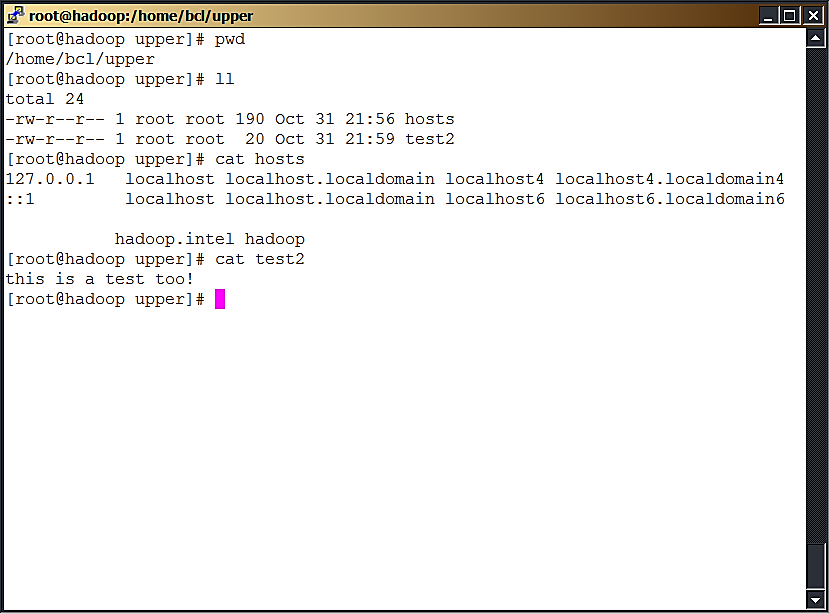
When I check the mounted filesystems, I see the following:



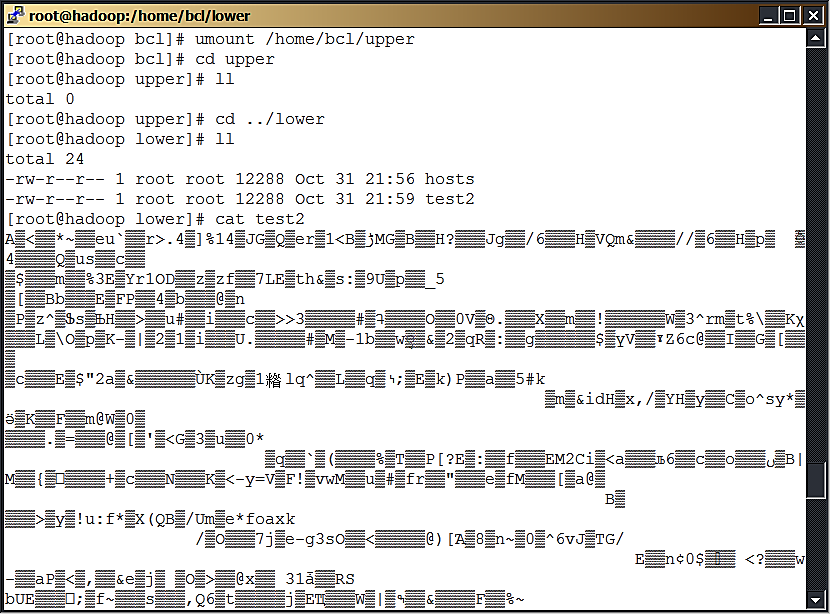
Now let me mount the encrypted ‘lower’ directory over the ‘upper’ directory. I was prompted for the key type and select #3 (passphrase). Next, I was prompted for a passphrase. I noted the passphrase since I would it every time I mount this encrypted file system. I selected the familiar encryption algorithm ‘aes’ (discussed in chapter 8) with a key 16 size of 16. Other options were left default. Please note, when you mount a filesystem for the first time, you will also receive a warning and a message indicating that the signature is attached to a file $HOME/.ecryptfs/sig-cache.txt:



You can observe that the files from ‘lower’ directory are now visible unencrypted in the ‘upper’ directory. Also, observe the size for these text files:

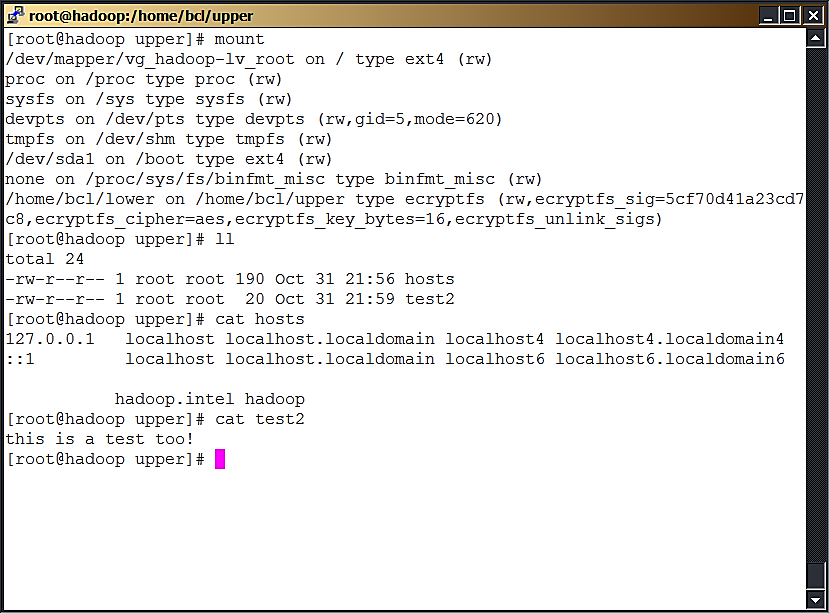


Let me unmount this filesystem and observe what happens:



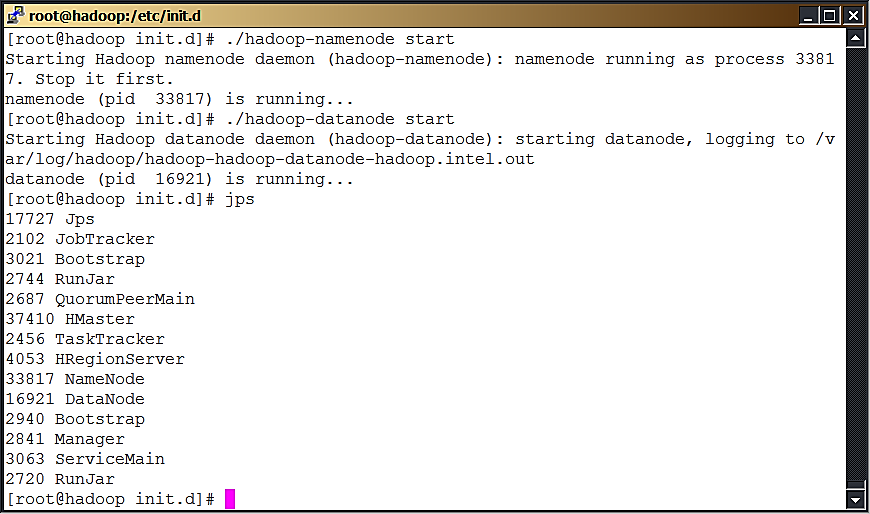
As you see, the files are not available in the umounted ‘upper’ directory and the size of these encrypted files has grown substantially within the ‘lower’ directory. If you now list the contents, all you can see is control characters. So, you have verified that eCryptfs works and encrypts as expected!

Let me mount the filesystem back and verify that my data is still intact:



So, the content within my text files is still intact.

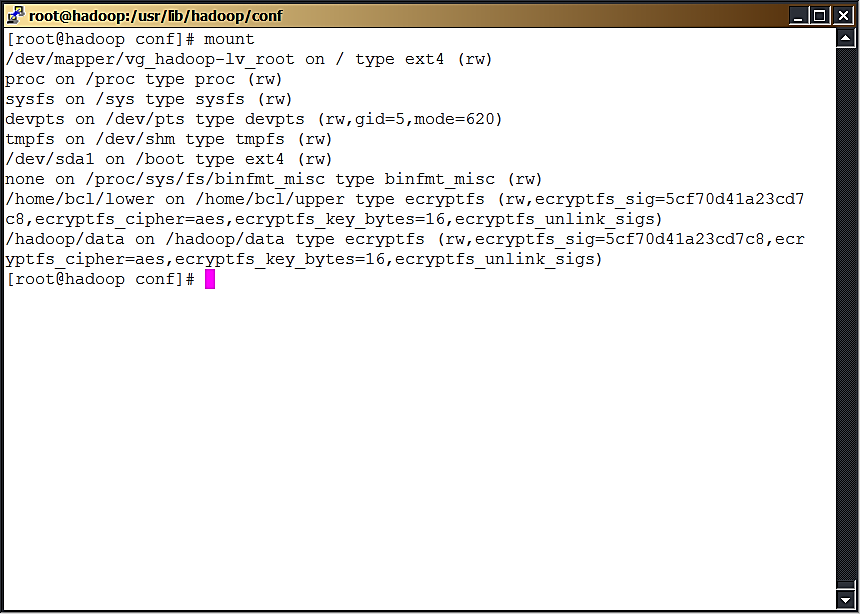
Next, let me demonstrate eCryptfs usage with Hadoop. First, start NameNode and DataNode services and verify they are running:



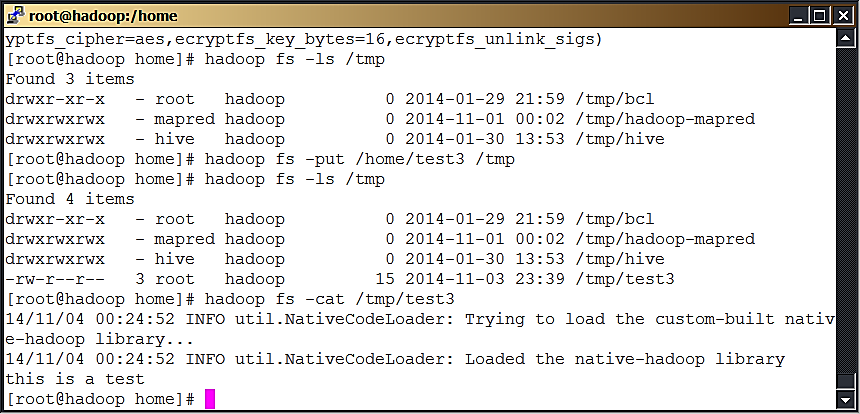
Next, locate the ‘hdfs-site.xml’ for your hadoop cluster (usually within $HADOOP\_INSTALL\_DIR/conf) and check the value for property “dfs.data.dir”. For my example it was “/hadoop/data”. So, need to encrypt directory “/hadoop/data” using eCryptfs, to implement encryption at rest for Hadoop. I used the following command to mount encrypted filesystem for “/hadoop/data” (‘#’ is the command prompt):

# mount -t ecryptfs /hadoop/data /hadoop/data

I was prompted for the key type and select #3 (passphrase). Next, I was prompted for a passphrase. I noted the passphrase and selected the encryption algorithm ‘aes’ with a key 16 size of 16. Other options were left default. I verified that the encrypted file system was mounted correctly:

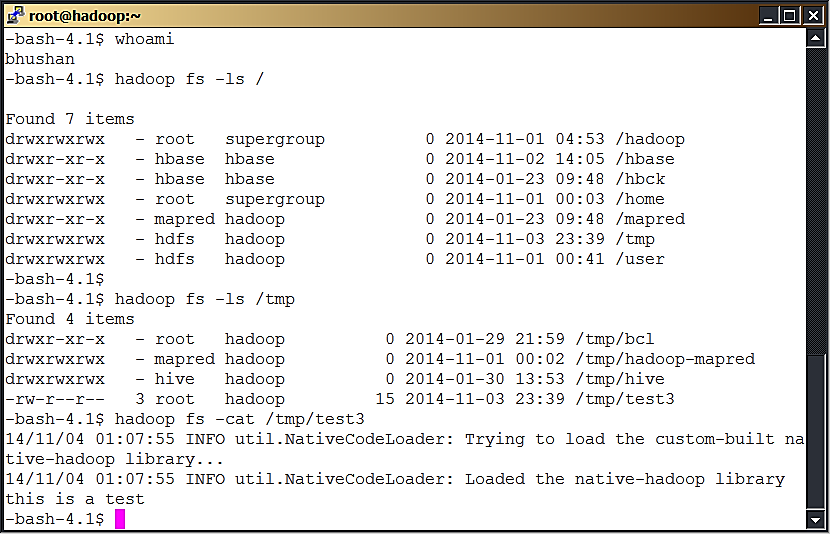


Next, I copied a text file (/home/test3) to the HDFS directory /tmp and listed the contents:

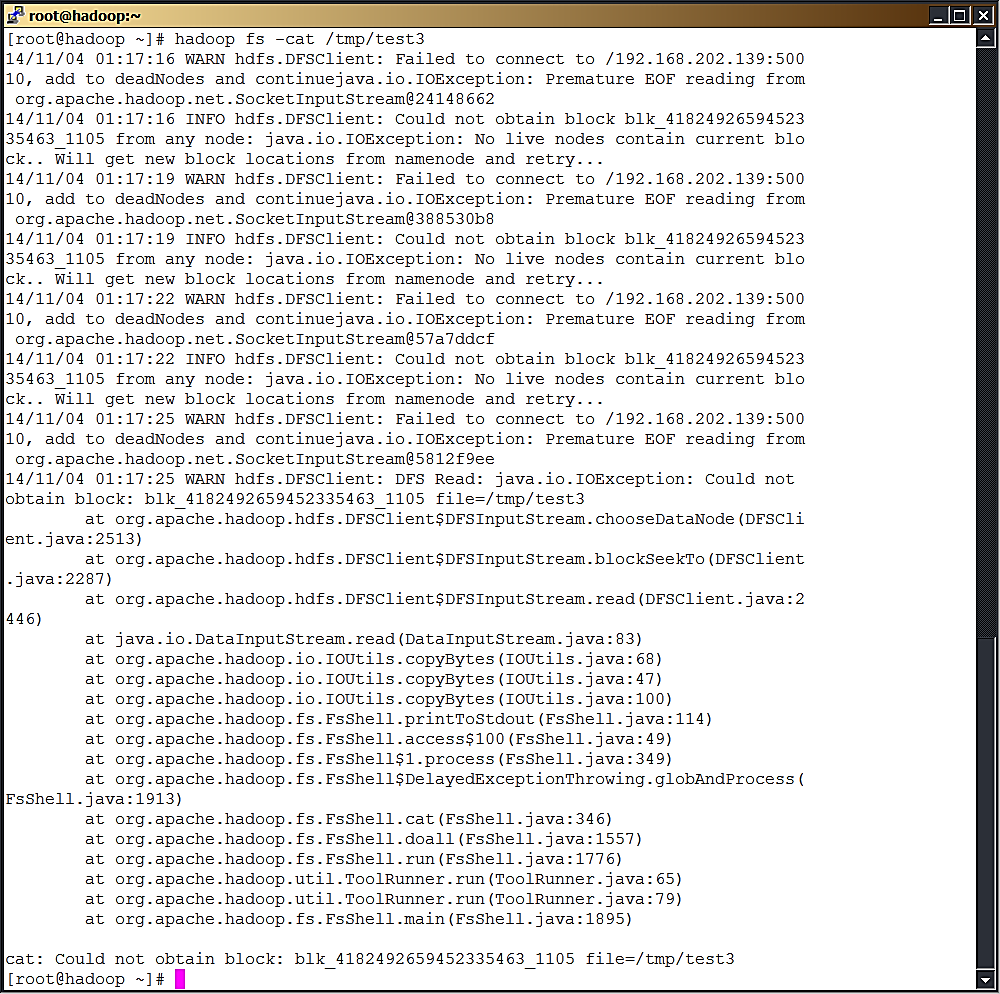


So, I have verified that I can read from as well as write to HDFS with the data directory encrypted!

Can I access this file as another user? Yes, once an encrypted file system is mounted, any authorized user with necessary access persmission can access the file. In this case, ‘test3’ has ‘644’ permissions. So, any user can read it. I created a user ‘bhushan’ and could read the file successfully:



What happens if I unmount encrypted file system? Well, you can’t access any HDFS files any more! You will see a lot of error messages followed by failure to access.



Please understand that the use of a passphrase in my example for demonstrating the concept (of using eCryptfs with Hadoop). For real world usage, you will need to use a keypair instead of a passphrase. Unfortunately, you can mount eCryptfs with a public key ony if you have public key support (pubkey) in your kernel.

In case your kernel has pubkey support, you first need to generate a public/private keypair. You need to use ecryptfs-manager (ecryptfs-manager is an application that manages keys and can generate new keys), follow the prompts to generate the keypair for the required key module, start the ecryptfsd daemon (ecryptfsd is a daemon that runs as the user performing file operations under the eCryptfs mount point. It manages public key operations in userspace on file open events. ecryptfsd only needs to be run when a mount is done with a public key module), and then specify the key module when mounting.

For example, for the OpenSSL key module (option 1 when mounting eCryptfs), assuming you created your key in /home/bclkey.pem and you want to do an encrypted mount on /hadoop/data, you will need to execute the following commands (‘#’ is the command prompt):

# ecryptfsd  
# mount -t ecryptfs -o key=openssl:keyfile=/home/bclkey.pem /hadoop/data /hadoop/data

In case your kernel doesn’t have pubkey support, you can save the passphrase to a file and control permissions to that file (as necessary). The passphrase file can then be used to mount eCryptfs.

I have deliberately used a generic open source operating system to demonstrate the usage of eCryptfs with Hadoop. You can easily procure an EMR cluster from AWS without server-side encryption or use of S3 storage. You can then follow the same process (as described in this section starting at page 1) and install packages ‘ecyptfs-utils’, ‘keyutils’ and ‘trousers’ to implement encryption at rest. Just remember to find out the correct “dfs.data.dir” for your EMR cluster and then encrypt it using eCryptfs!