Now we can also do the reverse we can decrypt some cipher text and get back the plain text. So this is the command. So aws kms decrypt. So it's a decrypt operation. I'm giving it a cipher text blob. So it's actual binary data, and it's going to be cipher text and I'm passing in the location of a topsecret.encrypted. Now, this is not based 64 encoded because it just expects binary data. So I'm giving it the raw file and I'm asking it for its output in text format. I'm querying it for plain text. I'm wanting the plain text component of the output, and its output will be base 64. So I need to pass it through to base 64 decode to get the actual plain text. So I press "Enter" and I'll get back my secret message. Now, one thing I want to draw attention to. **I did not have to specify the CMK to use to decrypt this data, and that's because whenever you have cipher text so encrypted data inside it there is a link back to the CMK that was used to encrypt that data. So you never have to specify the CMK to use to decrypt something.**

Now what you can also do with KMS is to use the reencrypt command. I want you to imagine the scenario where you're an administrator, maybe you're an S3 administrator, maybe it's a traditional file system. So you have access to all of the data let's say, on a normal flat file system, I want you to have that access because I want to make sure that you can administer the file system and make sure that that data gets backed up but what I don't want you to do is have access to the plain text version. All this data is encrypted and I don't want you ever seeing the sensitive let's say it's medical data in it's decrypted form. **What KMS allows you to do is run a re-encrypt operation, and with re-encrypt, you can provide it with the encrypted cipher text. You can give it a new key to use, and it will take that from you, re-encrypt it, and provide back to you the encrypted data using the new key and you will never see the plain text data, so that allows me to have role separation**.  
 **I can allow you to have access to do re-encrypt operations, so I want you to keep all of the data secure, and I want you to keep re-encrypting it with different keys to make sure that it's using new keys and they're rotated and there's no chance of any of this data being accessed but I never have to give you access to the plain text version of that data and that's one of the benefits of KMS It's capable of doing encrypt, decrypt, and re-encrypt operations.**

Now so far, I've talked **about encrypting and decrypting data of up to four kilobytes in size but there is a way that KMS can handle larger data, and that's by using data encryption keys.** So you can actually request to KMS to generate you a data encrypted key using a CMK and I want to show you the command that you would use to do that. So I'm going to switch back over to my terminal and clear the screen just to make it a little bit easier to see. So now I'm going to run this command to generate a data encryption key. So I'm going to use aws kms generate-data-key I'll need to specify the CMK to use. So --key-id and then the ID of the CMK and then --key-spec AES\_256 I'll do that and press "Enter." Now what I'm going to get back in return is two things. The first is a plain text version of this data encryption key and the second is an encrypted version of this data encryption key. So I could use this plain text version to encrypt a file locally and I can use this to encrypt a file of any size. Once I've done that encryption, I could discard this key. If I ever needed to decrypt that file, I could store this encrypted version along with the file on the same file system. In order to decrypt the file, I would need to pass this encrypted key back to KMS and say please decrypt it. Get back the plain text version of this data encryption key, and I would use that to decrypt the file. Now, that's called **envelope encryption**. **That's where you get one key and you generate another key and use that key to encrypt some data. So in order to decrypt that data, you need a few things you need the encrypted data itself. You need a copy of the encrypted data key. You need to decrypt that encrypted data key to get the raw data key. Then you need to use that to decrypt the object and so that's a multi step process** and that's actually how S3 functions if you use KMS. So for every object that you want to encrypt, S3 asks KMS for a brand new data encryption key. It encrypts the object, discards the plain text version of that data encryption key, and it stores the encrypted version along with the encrypted data and so that means that if you're an S3 administrator, you may have access to the encrypted object, but you can't decrypt it unless you have the ability to interact with the CMK and perform decrypt operations and because you can set the key policy you can define who is allowed to encrypt and decrypt with that key.

So **KMS has the concept of two types of uses on a key. You have the ability to interact and manage the key, but you also have a separate set of permissions, which allow you to perform cryptographic operations using that key.** So that's important. Now I'm going into a lot of detail, and you don't need to be aware of anywhere near this detail for the exam. I just want you prepared in case AWS do add any extra questions, which are using more detail than is currently present on the exam. What I know of the exam right now, you need to be aware of what KMS does so that you can create CMKs and integrate with other AWS products. You need to know that a CMK can only be used to encrypt or decrypt data up to four kilobytes in size. For anything bigger, you need to generate data encryption keys and those are used to encrypt data that's larger than that four kilobytes. Why this matters for the exam is that KMS is actually used for encryption and decryption for nearly all AWS products, and that includes RDS, DynamoDB, S3, and pretty much every other AWS product. If it uses encryption then behind the scenes its using KMS and with most products, you've got the ability to specify a specific CMK to use and that allows you to control the key in a really granular way using the key policy.

Now one last thing is that KMS has been enhanced recently and **the feature of custom key stores have been added. Now KMS up until very recently could only support FIPS 140-2 level 2. There was a separate product called Cloud HSM, which is a dedicated hardware encryption module and that supported FIPS 140-2 level 3 but now, with custom key stores, KMS is capable of interacting with a cloud HSM on your behalf. So there is an extra cost for this, but it now means that KMS can support level three as long as you configure this custom key store**. So I just want to be aware of that, not saying it's going to come up in the exam but I want you to be aware just in case. At this point that is everything I want to discover in this lesson about KMS. Go ahead, mark the lesson as complete and when you're ready, join me in the next.