M01-Ax010-CIS4204

Welcome to Penetration Testing and Ethical Hacking at the University of Florida. This is addendum Ax000 Key Encryption Methods, Key facts about keys arewhat we'll be discussing here.

What is it with these keys? Well, with most of the exercises that you complete there will be keys, one or more keys, that can be found. Sometimes they'll be in files. Sometimes they'll be in packets. Sometimes you'll have to look in different places. Sometimes you'll have to construct them. There are various methods that will be used to get you the keys, but they can be found. These keys are not readily transparent.

They're base64 encoded sixteen byte sequences, sixteen byte sequences. When you concatenate the base64 decoded strings together, they form a very long ciphertext that can be decrypted by you. If you're fortunate enough to find all of the keys, and many people can find them all, or if you can find enough of them, then you can decrypt the entire sequence, or enough of the sequence to be able to find a string of characters that will give you a passphrase that will provide you four points of leeway on the final exam. I already discussed leeway credit that makes up credit that you lose. And if you provide the wrong string—if you don't, provide the correct pass phrase that you find from the from the long decrypted string— then you'll lose two points. If you don't provide any string whatsoever, then your final exam grade does not change it's exactly what you earned.

So what do these keys look like? They look like this. There's the string, all uppercase "KEY" followed by three digits (they're sequentially numbered) followed by a colon followed by 22 characters from the set uppercase A to Z, lowercase a to z, 0 to 9 "+" and "/", followed by two equal signs. This particular key is the base64 encoded string "This isn't a key." And if you base64 decode the string you'll get sixteen bytes. Base64: it's the Multi-purpose Internet Mail Encoding (MIME) encoding that was originally used to encode files that were to be attached to mail, so you could encode sequences of eight bits, using only six bit characters. There's 64, yeah, 2 to the 6th, 64. So these could be reliably transmitted using available email implementations which at the time used 6 bits of data, a parity bit, and a stop bit in each 8 bit unit, each byte, that they would send. So it was aimed around the technology of the day.

What is AES? AES is the Advanced Encryption Standard, and this was based on a cipher called Rijndael, which was developed by Vincent Rijman and Joan Daemon, and it's a substitution permutation network cipher if you're familiar with that. If you're not, you know, it's some algorithm. Now, my choices for the parameters associated with the AES cipher are the following: The key size I chose is sixteen bytes, that is, 128 bits. So sixteen bytes: sixteen character size units. The cipher mode I chose is CBC. I'll talk about cipher modes in a minute. The initialization vector is the same size as the key that's sixteen bytes, and I chose this initialization vector. The "\n" is a new line, "\x" represent a Hex character: the two characters following that are the hex digits, associated with that character, etc. So that's the key I chose. Sorry, that's the initialization vector I chose.

I encrypt and decrypt, using Python3's AES class from the `Crypto.cipher` module `pycryptodome`. So if you want to do exactly what I do, and you need to install this on your own machine instead, uou want to `pip3 install pycryptodome`.

So I mentioned these modes. I use CBC mode. What are modes? Well, there's electronic codebook (ECB sucks, Look at Tux). It's a very simple structure. I'll talk about it a little bit more in a moment. Cipher Block Chaining, which is better, but not really great. And Galois Countermode is probably the best balance of security and performance that you can find. There are other modes as well. These are probably the most important.

How does ECB mode work? I already talked about how it was questionable. In ECB mode, each block is independently encrypted and decrypted, and the encryption and decryption are the same. You just replace the cipher text with the plain text, or the plain text with the ciphertext to encrypt or decrypt. I'm showing you the decryption, because that's what you'll be doing if you're looking at these keys. So for ECB mode, one provides sixteen bytes of ciphertext and a sixteen byte key to the algorithm and it produces sixteen bytes of the plain text. So if you have any of the blocks, then you can decipher that block. Also, if any blocks are identical, they can be deciphered to the *identical* plaintext. So if you have the key, you can recover the plaintext of any block.

CBC mode is a different approach to this that makes it a little bit more difficult to decipher. You can see it's that the diagram is more complicated. In this case, when you encrypt, the initialization vector is xored with the plain text before encrypting (you essentially flip this diagram upside down). To decipher the text. You provide the ciphertext and the key, and then you take the plain text, and you xor it with the initialization vector. For subsequent blocks, you do a different thing. After the first block, the preceding block's ciphertext is used as the initialization vector for that block. So what this means is, if you are missing a single block, you won't be able to recover the plain text for that block, and you also won't be able to recover the plain text for the following block.

Angecryption is a thing that Ange Albertini developed. So I use this idea a little bit. In addition to being the AES key, the encryption key that you're looking for to be able to decrypt or decipher the AES message It's also the cipher text associated with the initial key. So the message starts with the decrypted initial key. This method of providing the key and the initial decryption block was really suggested by Ange Albertini's Angecryption method. The important takeaway from Ange's talk is this. Ange was given a a challenge by someone, a decryption encryption challenge. And Ange really didn't know about encryption and decryption. And so it was a hard problem. But Ange realized something. You only need to know enough to get the problem solved. That's the world we live in today because we've lived so long as a species that there's too much information for any one person to know it all. Don't try to understand everything, just the important things. The hard part is figuring out what's important, not trying to understand everything.

But wait. I need this key! I need the key. Yeah. I never explicitly reveal the sixteen byte key that's used to decipher the long ciphertext. And that pass phrase is also the base64 decoded KEY000 which I never provide to you, so you'll never see it. So this presentation is very important. It's important because "The key is here." You can find it in this presentation. To quote the Mandalorian, "This is the way." That's just how we roll here. I'm not going to tell you what the key is. You need to find it. It's sixteen bytes. It's in here. It's in the presentation. As I mentioned before, the initialization vector is the one I provided earlier. It's complicated, but it's a vector. You can use that in a python program, whatever. It'll work right. You'll be able to do this. The rest, however, is up to you, soI'm not going to say good luck, because luck won't do it for you.

I will wish you good skill.