

Forum:[Hack Forums](#)

Thread: [Everything you need to know about Cryptography!](#)

Dvorak - May 8, 2014, 6:57 pm

Welcome to my guide on cryptography! Here I hope to educate willing people on the different subjects of cryptography. Anyways I hope you all enjoy the guide!

What will I be learning in this guide?

In this guide you'll be learning about the art of cryptography, which is the study of enciphering messages (I'll also be going over cryptanalysis which deals with breaking encryption). As you read this I expect that you'll pay close attention as to how things work as if you don't you won't leave this guide with anything useful. Enjoy! ?

What is cryptography?

Before we get into anything else, first we need to have a solid base of understanding what cryptography is and also have an understanding of cryptography's counterpart called cryptanalysis.

Cryptography

Cryptography is the study of enciphering stuff; whether it be communication, passwords, bomb recipes, or the latest porn that you downloaded. Essentially, if you have something that you don't want non-warranted peoples to be able to know you encrypt it. Cryptography is used by governments, military, journalists, banks, you (Even though you may not know it), spies, people who care about their security, etc. If you see where I'm going here cryptography is used EVERYWHERE, and little people know how it works. I'm going to be going over a bunch of different things that are all underneath the umbrella of cryptography, so if you still don't understand it don't worry!

Cryptanalysis

Cryptanalysis is the exact opposite of cryptography, cryptanalysis is the study of BREAKING encryption. This is done by governments, military, spies, black-hat hackers, and even good people who try and find holes in encryption schemes so that they can help patch up these holes therefore making the encryption scheme more secure. Cryptography and cryptanalysis go hand in hand, and without cryptanalysis cryptography would be no-where near as secure as it is today.

Some words/terms you need to know.

Before we start off, it's very important that you know what these words/terms mean so that you can understand this guide better.

Cryptography: The study of encryption and ciphers.

Cryptanalysis: The study of BREAKING encryption.

Plain-text: A normal string of characters that can be read normally, such as "Hello there everyone!" or "Attack the ship at 7:30 PM" (If you notice the latter string would most likely be used by a military).

Cipher-text: An ENCRYPTED string of characters. Here are the exact same strings shown above but now they're encrypted - "e5tZTBi6s3qn8+okIAOpPWSPqQ6VYxZxAid/IWLyx/E=" and "EEAeEu0H6DrdbfzU/IF9mTTAd2JQELADFoYC0EbnyVY=". So basically cipher-text is the stuff that

is encrypted and plain-text is the stuff that's unencrypted.

Key: This is a fixed length stream of characters. The key is what's used to lock/unlock encrypted items. The key can be different sizes depending on what type of algorithm you're using; for example, AES (Advanced encryption standard) 128 uses 128 bit long keys whereas AES 256 uses 256 bit long keys. Typically the longer keys you have the more secure the encryption will be but the encryption process can also take longer. Think of a key as you would in real life, without the key you can't decrypt a file that was encrypted with that key.

Algorithm (Can also be called a cipher): An encryption algorithm is the actual mathematical formula that is used to encrypt items. more on this later, just remember that the algorithm is like the machine that encrypts your data.

Password: This might get confusing here, now in different algorithms the password can be used for different things. I said earlier that the key is what's used to encrypt information, and that's right. But now you're probably wondering what the password does? The password as said earlier can be used for different things, in some algorithms the password is used to generate the same key every time, whereas in others it's used to "Unlock" the key so to speak. Even I get confused here sometimes so if somebody could provide a more robust answer to the difference between keys and password it'd be heavily appreciated.

Symmetric encryption: Symmetric encryption is an encryption algorithm that only uses one key, basically you encrypt your data with one key and when you wish to decrypt it you use the same key used to encrypt it. This is also commonly called "Private-key encryption".

Asymmetric encryption: Asymmetric encryption is an encryption algorithm that uses two keys, so if I were to use an asymmetric encryption algorithm to encrypt my data one key could encrypt it but not decrypt it, whereas the other key could decrypt it but not encrypt it. This is also commonly called "Public-key encryption".

Hashing: Hashing is different from encryption, with an encryption algorithm you can encrypt items but you can also decrypt them, whereas with a hashing algorithm you can only get the "Encrypted" version but you can't reverse it and get the original data again. More on this later.

Brute-forcing: Brute forcing is the act of BREAKING encryption by trying to use EVERY SINGLE encryption key until it finally breaks the encryption algorithm. As said earlier encryption keys can be different lengths depending on the encryption algorithm. So for example if an attacker were to try to crack a document encrypted with AES-256 they'd have to try every single possible 256 bit key combination. The reason AES-256 is considered secure is because for an attacker to do this it could take the amount of time it would take until the universe expends all of it's energy, essentially making a brute-force attack unfeasible.

Word-list: A word list is a .txt file that has a pre-computed list of passwords. These lists can differentiate from each other, as some have all possible passwords up to a certain amount of characters long whereas others contain the most commonly used passwords (Get that list [HERE](#)). These word lists are most commonly used in brute-forcing schemes to try and crack into peoples accounts.

Home-brew encryption: Home brew encryption is an encryption algorithm that is made by someone who isn't well known for creating encryption algorithms. Home brew encryption is normally advertised as secure when in reality it's easily cracked. Home brew encryption is also normally untested for security issues. **NEVER USE HOME BREW ENCRYPTION FOR ACTUAL DATA PROTECTION!**

Snake-oil encryption: Snake oil encryption is a typically “Commercialized” encryption algorithm. This means that it’s relatively untested, the companies who sell it make broad and non-honest statements about it, the source code isn’t publicly released, etc. Basically snake oil encryption are encryption algorithms that lie about their actual capabilities and only want to make a quick buck off of you. NEVER USE SNAKE OIL ENCRYPTION!

Closed source: This means that the encryption algorithm’s source code isn’t publicly released, therefore cryptanalysis isn’t able to be done on the source code and security issues aren’t able to be fixed. NEVER USE A CLOSED SOURCE ENCRYPTION ALGORITHM!

Open source: This means that the encryption algorithms source code IS publicly released, therefore cryptanalysis IS able to be conducted on the algorithm and security issues can be patched quickly. ALWAYS USE THESE KINDS OF ENCRYPTION ALGORITHMS!

Now that you understand what these words/terms mean we can continue!

How do keys and algorithms work together?

To be able to get encrypted data you need three things.

1. The data you want to encrypt
2. An encryption algorithm
3. An encryption key that is able to be used with that algorithm

Once you have those three things you can take the data you wish to encrypt, send it into the encryption algorithm with the key, and the output will be encrypted text. If you want to DECRYPT that data, you send the encrypted text into the decryption algorithm with the key and the original data will pop out. If we were to use anything other than the key that was used to originally encrypt the data, it wouldn’t work.

Here’s an example of a very simple cipher that would take a number, and encrypt it.

$\text{Cipher-text} = \text{plain-text} + \text{key}$

So if my plain-text was the number 50 and my key was the number 100 my cipher-text would be 150. Here’s an example of the decryption function for this cipher.

$\text{plain-text} = \text{cipher-text} - \text{key}$

As you can see above if the cipher-text is 150, and the key is 100, you’ll get the original number of 50. The only requirements are that you have the cipher-text and the correct key.

In real life these calculations are MUCH more complex and much more difficult to crack. [HERE](#) is an example of another cipher and how it works with the actual alphabet instead of just numbers like my above example.

Now that I know how symmetric encryption works how does asymmetric encryption work?

In my above example you only needed one key for the encryption function and the decryption function, whereas if you read above you know that asymmetric encryption involves two keys.

Asymmetric encryption is when two keys are used for encryption/decryption, and here’s an example of it in real life.

Let's say you own a bank. When users connect to your bank's website to read things such as recent transactions you want their connection to be encrypted so that bad guys can't intercept the connection and collect all of that data themselves. To do this you'd use asymmetric encryption. You the bank would create a key-pair, this key-pair is a group of two keys that are mathematically created so that if one of the keys encrypts data only the other key can decrypt it. We're going to name one of these keys the "Private key" and the other key the "public key". The private key is the one that ONLY the bank knows, whereas the public key may be known by everyone. A client who is going to login to your website would encrypt their connection with the public key, now when they connect to your bank's website if anyone tries to read the connection all they'll see is gibberish. When the connection arrives at the bank's website it'll be decrypted by the private key, as only the private key can decrypt it. This happens in real life all of the time and that's why I said earlier that you might be using encryption and not even know it.

If you want to learn more about asymmetric encryption I advise you to [HERE](#)

What's hashing?

Hashing... Many peoples get encryption and hashing intermixed, and I'm here to tell you the difference.

As seen earlier encryption can and is meant to be reversed; provided you use the correct key. Hashing is NOT meant to be reversed, you enter data (Such as a password) and it gives you an output. This output CAN'T be used to get the original input again. An example of a hashing algorithm is shown below.

$$\text{\$hash} = \text{\$password} + \text{\$password}$$

As you can see above a password is added to itself and the sum is the hash. The hash isn't meant to be reversed (Even though it could quite simply with a brute force attack). Obviously in real life hashing algorithms are much more complex and wouldn't be able to be cracked as easily as my example would be.

A real life example of hashing is when a user enters their password to login to a website. The password is hashed by a secure hashing algorithm, and the hash is then sent to the website. If the hash matches the hash that corresponds to the account the user wants to login to, they gain access. Now if an attacker were able to grab the database of all of the website's accounts they won't be able to find out the passwords because all they'll see are the hashes of these passwords, and the hashes can't be reversed.

What are salts and how do they relate to hashing?

To explain this I have to explain rainbow tables. Rainbow tables are word lists full of hashes, these hashes are derived from commonly used passwords. So if an attacker grabs a database full of account information from a website and the passwords are hashed, the attacker can compare those hashes to the hashes in his/her rainbow table. If any of the hashes in the rainbow table match the hashes in the database, then the attacker knows that password.

Now since many peoples use commonly used passwords, this is very dangerous. A salt is just an added piece of data that's used in the hashing process, to explain better let me use my example hashing algorithm above and modify it to use a salt.

$$\text{\$hash} = \text{\$password} + \text{\$password} + \text{\$salt}$$

The salt could be anything, from the URL of the website to a users favorite animal (Ever wonder

why websites ask you that question whenever you login?). All the salt does is add in additional data to the hashing process so that rainbow tables don't work. Go [HERE](#) for more information on hashing, and go [HERE](#) for more information on salting.

What does the difference in key sizes mean?

The key, which is used with an encryption algorithm, can be different sizes depending on which size the encryption algorithm needs. For example AES 128 needs 128 bit keys, whereas AES 256 needs 256 bit keys. In general the longer a key gets the more secure the cipher text becomes, but at the same time a longer key size means a longer encryption/decryption time frame. For example, if I were to use AES 256 over AES 128 AES 256 would technically have a more secure key size because the key is 128 times more difficult to crack than AES 128 (An example of AES 128's security is it would take the age of the universe to crack, so AES 256 would take 128 times that length. Unnecessary but people like the thought of that.). For another explanation of key sizes I direct you [HERE](#)

What's better? Algorithm or key size?

When peoples want to be as secure as possible, lots of them look explicitly at key size, this is the WRONG thing to do. Of course key size is important, but it's not as important as the algorithm itself. If a poorly implemented encryption algorithm uses a massive key size, it will still be easily cracked because of how the algorithm hales security holes in it. Essentially, the key size is important, but not as important as the algorithm itself.

Randomness

Nothing, absolutely nothing is truly random. Many people will say that atmospheric noise is truly random, but it isn't. How does this relate to cryptography you might ask? Well keys need to be as random as possible otherwise the keys would be easy to guess. When something isn't truly random (Which nothing is) it's called pseudo-random. Pseudo-random generators are algorithms that generate as seemingly random information as possible. They can use mouse movements by the user, hard drive movements, atmospheric noise, etc. These algorithms are used because even though they aren't truly random, they're random enough to be considered secure.

Digital signatures

In real life, if you want to prove a document was made or authorized by you you'd use your signature. Computers do the same thing. Remember asymmetric encryption? Well asymmetric encryption is also used to digitally sign data so that you know who it was made/verified by. How does this work?

Let's say you want to digitally sign a document that you're sending to an important friend, you would generate a key pair, and use the private key to encrypt the document and you'd give the public key to your friend. Now when you send the document to them if they're able to decrypt the document using your public key, it proves that you wrote it. Pretty nifty eh? Go [HERE](#) to learn more about digital signatures.

Additional websites

Here are some websites that you can visit to learn more!

https://www.owasp.org/index.php/Guide_to_Cryptography

<http://epictuts.blogspot.com/2011/06/beginners-guide-to-cryptography.html>

<http://msdn.microsoft.com/en-us/library/aa480359.aspx>

<http://www.cryptographyworld.com/>

A good book to read that involves a lot of cryptographic content

I just finished reading [THIS](#) book that you can download for free, it contains a lot of information about cryptography and has an awesome story as well. Check it out!

Different algorithms

Symmetric encryption algorithms

1. AES (128, 192, and 256 bit keys)
2. Serpent (128, 192, and 256 bit keys)
3. Twofish (128 - 256 bit keys)
4. Blowfish (32 - 448 bit keys)

Asymmetric encryption algorithms

1. PGP - Pretty good privacy (512, 2048, 4096 bit keys)
2. RSA (1024 - 4096 bit keys)

Hashing algorithms

1. Whirlpool
2. SHA-2/3
3. ripemd-160
4. SHA-512

Helpful programs for cryptography

TrueCrypt

TrueCrypt is an open source directory encryption program that lets you pick from different encryption algorithms and hashing algorithms. Very secure and fairly easy to use. You can find it [HERE](#).

KeePass

KeePass is an awesome way to keep all of your passwords safe for whenever you want to use them. It's open source and you can get it [HERE](#).

Closing statement and additional information

Thanks for reading! If you have ANY questions feel free to ask me and I'll gladly answer them and if I can't I'll point you to a place that can. Peace out! ?

Hitch - May 8, 2014, 8:29 pm

can you learn me how to make crypter ?

Corwin - May 8, 2014, 8:30 pm

Very nice thread man, very well explained.

crazyycast13 - May 8, 2014, 9:10 pm

>that feel when you have been working on a tutorial for a while, then some random person comes out of nowhere

Ehh looks pretty complete though.

Dvorak - May 8, 2014, 9:29 pm

(May 8, 2014 09:10 PM)crazyycast13 Wrote: [CITE](#) >that feel when you have been working on a tutorial for a while, then some random person comes out of nowhere

Ehh looks pretty complete though.

Hello, I'm very sorry if I made this guide while you were making one similar to it. Sorry again ?

crazyycast13 - May 8, 2014, 9:33 pm

(May 8, 2014 09:29 PM)James R. Bennings Wrote: [\[CITE\]](#) Hello, I'm very sorry if I made this guide while you were making one similar to it. Sorry again ?

Nah man, it's okay. It's a good starter guide, I'm going to link to this one as suggested reading. I'm just going to go more in depth about crypto.

Dvorak - September 30, 2014, 2:29 pm

Sadly I'm unable to update my original post due to being past the time limit.

I'm here to update you with information of another crypto program called [Tomb](#). Tomb is a shell script that sadly only works on Linux (Though they may be creating a version for Windows) but it has MANY awesome features such as key and tomb-file separation, image stenography, and even the ability to transform your key into a QR code you may print! It's awesome and if you use Linux I HIGHLY recommend you try it out.

BlacksmithSec - October 1, 2014, 3:23 am

tut i notice that the op has gone to the effort to write such a long and detailed piece, thats clearly had a reasonable amount of time and effort invested in it, only for the the first comment to be the illiterate begging:

"can you learn me how to make crypter ?"

Good work op :)

asmy - October 1, 2014, 9:54 am

thanks can you learn me how to make crypter ?

rweed1 - February 4, 2015, 12:58 pm

Very detailed thanks for the info man!