|  |
| --- |
|  |
| Capstone Experience  IST 894 |
| Lab 9 – Cryptography  Scott Finlon |

Table of Contents

[1. General Context 3](#_Toc86961715)

[2. Technical Context 4](#_Toc86961716)

[3. Solution 6](#_Toc86961717)

[References 13](#_Toc86961718)

# General Context

In this lab we utilize two different tools to do some symmetric and asymmetric encryption. Both scenarios can and are used in secure messaging. First, we use an application called ‘ccrypt’ which utilizes symmetric encryption to encrypt and decrypt some text files, this means that it uses the same key to both perform encryption and decryption functions. Then we use GNU Privacy Guard (GPG) which utilizes asymmetric encryption to sign, encrypt, and sign and encrypt some text files. Asymmetric key encryption means that every user has a private and a public key and you encrypt with one and decrypt with the other.

The lab states that ccrypt was created to replace bcrypt, when it was designed as a replacement for the standard Unix crypt utility, which is notorious for using a very weak encryption algorithm. Ccrypt is based on the Rijndael cipher, which is the U.S. government’s chosen candidate for the Advanced Encryption Standard (Ccrypt | Kali Linux Tools, n.d.). This cipher is believed to provide very strong security, as it utilizes 256-bit encryption which is still today considered unbreakable. 256-bit encryption means that the key that is used to encrypt the data is 256 bits long, what this means in plain English is it can have 2^256 unique combinations which is a 78-digit long number. Because of the sheer number of possibilities, and the state of current computing power, it’s not feasible to attempt to brute force any 256-bit key (ClickSSL, n.d.). One of the biggest weaknesses of symmetric encryption is because the same key is used to encrypt and decrypt a file, anyone who obtains the key can view the message. There is no way to ensure that only certain recipients can view it, and there is no way to

GPG is a free and open-source implementation of the OpenPGP standard. One of the largest benefits of utilizing asymmetric key encryption is that while symmetric key encryption can provide confidentiality, it can not provide integrity of knowing who sent a message and being able to prove that the message has not been tampered with. Asymmetric key encryption allows a user to digitally sign a message, which can then be verified by the receiver that it’s not been modified in any way and the signing process can happen with or without encryption. Due to this, GPG encryption is often utilized in email settings to prove that you are the one who sent an email, but also can encrypt it in a way that only the intended recipient can open it.

# Technical Context

There are distinct advantages and disadvantages to both symmetric and asymmetric encryption. The best way to know which method to choose is to look at your individual use case. We can compare each process in key sharing, message security and validation, speed, and what happens if and when a key becomes compromised.

In order to share an encrypted message, the recipient needs to be able to decrypt it. Since symmetric key uses the same one key, it requires you to securely get the that key to everyone that you want to send messages to. Asymmetric encryption on the other hand has every user generate their own public and private key, so the total number of keys goes up exponentially for each person in the message change. Say for example you want to send a message to four people, with symmetric encryption you all only need one key but there is nothing that will alert you if someone else obtains the key and intercepts the message and changes it and re-encrypts it before it gets to the recipients. For that same four person message with asymmetric keys, you have five total people including the sender multiplied by two keys each for ten total keys that are utilized in the encrypting and decrypting that same message. Due to this, asymmetric encryption does not scale well, but the integrity and authentication that it can provide can make it worthwhile to use it.

As far as speed goes, symmetric encryption is faster than asymmetric, so symmetric is used more in streaming communications and asymmetric used in less time sensitive ones. Symmetric is faster because it sacrifices security for speed, while asymmetric sacrifices speed for security (Daniel, n.d.). Ccrypt uses a 256-bit key, where GPG can use key lengths of up to 4096 bits. The massive difference in key size is the largest factor in the speed differences.

Finally, what to do if a key becomes compromised. In asymmetric encryption, if you need to change or rotate a key there is no way to easily let everyone know that the key should no longer be used. You need to contact everyone and re-distribute a new key securely. GPG and other asymmetric encryption methods can utilize Public Key Infrastructure (PKI) to establish a Web of Trust (WoT) (Building Your Web of Trust, n.d.) which means that there are servers out there where people can post their public keys along with signatures from other peoples keys that is used to indicate that the key is trusted. Asymmetric keys are meant to be validated in person at key signing parties or other events, and you should only sign a key of someone that you personally know and have validated. Not doing these steps makes the WoT not reliable. Utilizing PKI makes key revocation much easier, you can just run a few simple commands to generate a revocation key and then upload it to a public key server (Revoking a GPG Key, n.d.). Once it’s been uploaded to a public server, other key infrastructure will see that that key is no longer valid and warn you not to trust it.

# Solution

Since ccrypt isn’t installed on Kali, we need to run `sudo apt-get update` and then `sudo apt-get install ccrypt`. We then use `wget` to pull down a text file for utilization in this lab. We run `ccrypt --help` to see the available commands and options, and then run `ccrypt -e textfile1.txt` to encrypt textfile1.txt. This process creates a file called ‘textfile1.txt.cpt’ where the .cpt extension indicates that it’s cipher text and not plain text anymore. Finally, we run `ccrypt -d textfile1.txt.cpt` to decrypt the .cpt file back into plain text.

A picture containing text, plaque

Description automatically generated

Figure 1 - ccrypt help menu

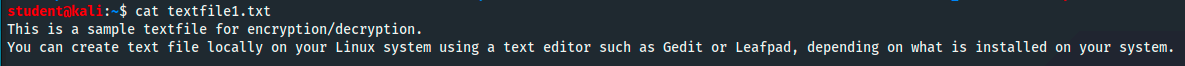


Figure 2 - Output of original plain text file

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3 - Encrypt the file and then show cipher text

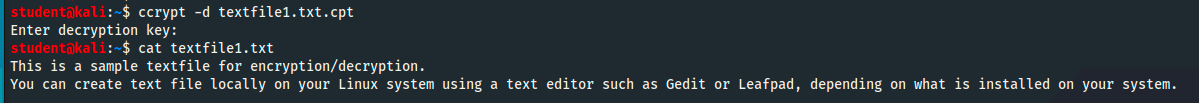


Figure 4 - Decrypt file and show plain text output

To begin using GPG, we need to create a key pair. We run `gpg --gen-key` and add in our name and email address and it saves the keys to our private GPG keyring. We use ‘wget’ to pull down a second text file for use. We then run `gpg -e -r [smf261@psu.edu](mailto:smf261@psu.edu) textfile2.txt` where -e tells it to encrypt and -r is the recipient for the message. We then decrypt the file with `gpg -d textfile2.txt.gpg`. We then export our public key from the key ring by running `gpg --export -a [smf261@psu.edu](mailto:smf261@psu.edu) > public.key` so that we can share it either directly with other users or to be placed on a key server. If we are given another users public key, we import it to our key ring with `gpg --import public.key`, and we can then list the keys in our key ring with `gpg --list-keys`.

Text

Description automatically generated

Figure 5 - Generate a new public/private key pair

Text

Description automatically generated

Figure 6 - Read file, encrypt file, and show cipher text

Text

Description automatically generated

Figure 7 - Remove original text file and then decrypt cipher text

Text

Description automatically generated

Figure 8 - Export my public key from my key ring

Text

Description automatically generated

Figure 9 - Import a key to my key ring

Text

Description automatically generated

Figure 10 - List all keys in key ring

**We digitally sign a file with `gpg --sign textfile2.txt` which signs the file and them compresses it and creates a binary file called ‘textfile2.txt.gpg’. The signature is validated with `gpg --verify textfile2.txt.gpg`. We can make a clear text signature with `gpg --clear-sign textfile2.txt` which signs the file but doesn’t otherwise modify the file and creates a plain text file and signature called ‘textfile2.txt.asc’ which can be verified in the same way `gpg --verify textfile2.txt.asc`. There is a third method for signing a document that creates a detached signature, where both the document and the detached signature are needed to verify that the signature integrity is in tact, this is done with `gpg --detach-sign textfile2.txt` which creates the detached file in ‘textfile2.txt.sig’.**

Text

Description automatically generated

Figure 11 - GPG basic digital signature

Text

Description automatically generated

Figure 12 - GPG clear-signed signature

Text

Description automatically generated

Figure 13 - GPG detached signature

Finally, we can go to a key server and search for my last name ‘finlon’ and find various GPG keys that I’ve used since 2010. They have number keys, subkeys, and signatures that create a Web of Trust with my coworkers and colleagues.

A screenshot of a computer

Description automatically generated with low confidence

Figure 14 - GPG key with subkeys for different email address with signatures

Graphical user interface

Description automatically generated

Figure 15 - GPG key on key server with signatures and signed image

# ****References****

*Building your web of trust*. (n.d.). Retrieved November 4, 2021, from <https://www.gnupg.org/gph/en/manual/x547.html>

*Ccrypt | Kali Linux Tools*. (n.d.). Kali Linux. Retrieved November 4, 2021, from <https://www.kali.org/tools/ccrypt/>

ClickSSL. (n.d.). 256 Bit Encryption: Is AES-256 Bit Encryption Safe in Modern Times? *ClickSSL Blog - Information about SSL Certificates & Infosec*. Retrieved November 4, 2021, from <https://www.clickssl.net/blog/256-bit-encryption>

Daniel, B. (n.d.). *Symmetric vs. Asymmetric Encryption: What’s the Difference?* Retrieved November 4, 2021, from <https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption>

*Peter Selinger: Ccrypt—Frequently Asked Questions*. (n.d.). Retrieved November 4, 2021, from <http://ccrypt.sourceforge.net/faq.html#sec>

Project, T. P. of the G. (2021, October 13). *The GNU Privacy Guard*. The GnuPG Project. <https://gnupg.org/>

*Revoking a GPG key: Hackdiary*. (n.d.). Retrieved November 4, 2021, from <https://www.hackdiary.com/2004/01/18/revoking-a-gpg-key/>

*Search results for “finlon.”* (n.d.). Retrieved November 4, 2021, from <https://keyserver.ubuntu.com/pks/lookup?search=finlon&fingerprint=on&op=index>