## Linux -- Using GnuPG with SSH for Encryption Automation

GnuPG is a set of encryption tools, funded mostly by the German government that allows you to safely send messages and documents over the Internet. It does so by using a variety of methods, the most common of which uses a key pair methodology that you create. There are currently two variants of the gpg tool called gpg and gpg2. The difference between the two is gpg2 has a different architecture than gpg in that it splits up the functionally into several modules. The gpg command is smaller and does not depend on other modules at run and build time.

How the GunPG project software works is much akin to how your house or car keys work to unlock your house's front door or your car door. You have keys on your key chain that you use to gain access to your residence or vehicle. You hope that anyone without your keys will not be able to break into your car or house. How far you want to go in time and expense to protect these assets is up to you to determine. For example, you can add alarm systems, unbreakable glass, and keyed lug nuts on your tires, or pull down covers that conceal what is in the back of your SUV. GnuPG works much the same way in that you can exchange data with varying levels of security, such as denying crackers access to data without the proper keyfiles, passwords or passphrases. If you make a copy of the keys to your house or car at the local hardware store and give it to a trusted relative, friend or acquaintance, you have given them access to do what they may want with your assets. They can rob your house or steal your car with impunity. In GnuPG terms, this is called your digital private key, which should never be revealed or distributed except under the most trusting of circumstances. This is a very simplistic description of a somewhat complex tool, but I hope you can understand that the concepts behind this software are not all that difficult or different from your everyday life. Our entertainment industry seems to like to dramatize this technology into something of science fiction that only geeks, government spooks and crackers can understand. I'm hoping to show you that is not the case.

With GunPG, you would use $ gpg --gen-key command to generate a pair of keys consisting of a public key and a private key that is encrypted to a passphrase that only you know. Your private key is the newly installed locks on your house front door. The public key that you generated can now be distributed to your partners so that you can exchange data back and forth privately and securely. You can distribute your public key to the whole world through public key services or limit its distribution to a few select friends, relatives and/or business partners. The public key that you freely give out could be viewed much like what we see in the movies when the evil villain is trying to launch a nuclear missile to destroy a city. The movies depict the silo operators as two trusted officers with two separate and differing keys that must be turned at the same time for the missile launch to occur. If one of the keys were lost or stolen, it would be useless without the other key. This analogy is not perfect, but in our case we only have one private key to protect; the other public key we can freely distribute. Our private key is the only way that crackers can gain access to our encrypted data. If our private key is ever compromised, we can always generate a new key pair. This would be like changing the locks on your house doors because someone has a copy of your house key that presents a threat. Just as you should never give your new key to anyone that you don't fully trust, it is very important that your new digital private key be kept secure and never shared with anyone.

If/when you give others your public key(s), it is important that you verify that they have ***your*** public key. Some middleman may have intercepted your key in route and substituted one of their own. For example, someone intercepted (stole) a credit card application from your mailbox, changed the address on the application and is now using that credit card with impunity. Since you will never even see a bill and no one at the credit card company verified that you received the application that they sent to you, an Identity Thief can now do some REAL damage to your credit. I hope this example explains how important it is to verify an exchange of keys. This also shows how companies are not held accountable for setting you up to have your identity stolen. You will have to deal with the aftermath of this disaster, not the credit card company.

A key exchange is not as hard to do as it is in the movies where an armed escort is needed. It is done by examining what is called the 40-digit "fingerprint" of the keys that you generate and exchange. This fingerprint is unique to anywhere else in the world, i.e. no other person will have a certificate with the same fingerprint. Most times only 8 digits are needed to verify your public key, as it is quite unlikely that the same 8-digit sequence would occur twice anywhere in world. For this reason the key ID or fingerprint is often only displayed as the last 8 digits. This fingerprint identifies the identity of the certificate, which is much like how the fingerprint of a person uniquely identifies a person.

Any form of simple communication can be used between you and your public key holder to verify that they have your public key, such as a private phone call. Your partner should do the same with you and once you have your partner's public key, you can then add it to a file called your public keyring. Once keys are verified you need some sort of mechanism to show the extents to which you went through to verify the keys and prove you or they are who you say you are. This is a very important step in the process, sometimes skipped by corporate or government IT employees, but this is critical to the verification and signing of your digital keys.

Verifying and signing a public key is an added and necessary step to enable the safe exchange of data between you and your partners. This process allows you and your partners to keep track of all the keys that you have all verified. It also a way of establishing a "Web of Trust" so that others can verify their keys based on your recommendations. We will get into the technical details, importance and further description of this process soon.

To securely send data to your partners securely, you encrypt your data to your private key and your partner's public key and send the encrypted data to your partner. Because the document is encrypted to their public key, your partner can decrypt the document using their private key. When your partner sends you a document they will encrypt it to your public key and you will then use your private key to decrypt the document. When your partner gives you their public key you should sign the key on your personal keyrings to show that the key has been verified.

If the data is a document, there are varieties of protocols that can be used to transmit documents to and from your partners. In 2006, most U.S. corporations were still using the insecure FTP protocol to transfer files. In 2013, when working on setting up my small business I have had to interact with other small businesses while writing this this book. What I found is that these million/billion dollar business were amazingly still using archaic insecure third party software and the insecure FTP protocol to exchange sensitive data with their customers. FTP was invented in 1971 and received some security updates in 1998. It has passive mode communication connection, which is/was somewhat secure and which I scripted for in 2006, but you must move beyond this outdated technology. Any business partners such as legal firms, printing companies, etc. that require you to use this form of file transfer is not someone that you want to continue doing business with, for security reasons. You are putting all your data at risk. This is akin to using Windows XP, which we previously discussed, which we have to assume that a large percentage of the world is spying on all XP Internet interaction with your partners. As I write this there seems to be little investment in cyber security education by SBO or large entities in regards to training their IT personnel how they should be interacting with and using the infected Internet. I hope you now understand that you must include how seriously your partners take cyber security in your business decisions.

For example, when working with one partner, I suggested they set up an SSH server to replace their FTP server. We will cover SSH in [Chapter 9](#_-_Hardening_security). I'm sad to say I was met with indifference, which I found disturbing, so I limited my data exchanges and business with them. I even offered my services at a bargain in the hope of benefiting us both, but again was met with indifference. Unfortunately, I had to risk my work on this book by connecting to them via FTP to conduct business. Many SBOs also use paid third party services, who they think are employing proper security techniques, but the reality is they rarely do. The files I transferred were not even encrypted, which demonstrates how some HCUs/SBOs do not even have a very simple/basic understanding of cyber security.

|  |  |
| --- | --- |
| **Story** | Back in 2006 while working for multi-billion dollar corporations, all of the partners that we interacted with still used the insecure FTP protocol to exchange data. I managed all levels of file encryption for a major corporation using FTP. Because FTP is an insecure file transfer protocol, I had to devise methods for keeping our corporation's encrypted data secure. I certainly did not want to be the person responsible for the stolen identities of thousands of employees' and corporate customers' identities being stolen.  Imagine the liability I might face, as well as the guilt that I would feel, for not exploring and developing every means possible to make sure a data breach never happened. This was a huge burden on my conscience and one that I took very seriously. The corporate answer back then was to encrypt files with PGP and then send them via FTP to our partner's computer servers. However, the company had me automate everything so we could cut back on staff. To accommodate management I invented some crazy scripts and algorithms in trying to protect their data and automate just about everything else. Although, in automating everything, someone could easily copy the scripts that I was using to communicate with our customers/partners, who could then compromise the corporation's data. Those scripts had the password to our private encryption key, thus giving anyone with access the power to unencrypt everything. This information could be worth millions to identity thieves. Even though it kept me employed for at least another year by automating everything, I didn’t feel comfortable doing what they were asking me to do, which was to use an insecure unencrypted file transfer protocol and scripting encryption passwords. However, when you accept a paycheck, you do as you are told. Any cracker or corrupt corporate administrator gaining access to those automation scripts would have the keys to all of the corporations' data.  I also felt it might prove dangerous to bring in low-paid, un-vetted foreign workers into our payroll department that would handle all of our confidential data, thus giving them access to all of our sensitive corporate data, but that was not my call. I felt that these workers might have other agendas or be working for other entities that did not have the corporation's best interest in mind (spies). Corporate espionage is a known problem that gets very little attention in our courts or media. When/if you are placed in this position, you have to trust in your talent and abilities to move on to bigger, and better things; and I later did, so I documented everything as best I could for the new foreign workers who would later perform my job. Conversely, I knew that by doing so I might be sacrificing the private information of all the customers, employees, and affiliates of the corporation thus revealing all the data in of all our private corporate databases to people who probably should not have that access. Cyber espionage\* is real and can impact corporations and their employees in painful ways.  I give this example not as self-aggrandizement but to show you how your private information is made public that you later have to deal with, as well as how there are no laws to penalize corporations for this cost-cutting activity that result in a lack of internal cyber security. Corporations seem to have lost the perspective that the lifeblood of any organization is the people who work for it. We all have to determine just how far we want to allow this privacy invasion to continue until proper laws are passed and cyber security is taken seriously. This is akin to how military cutbacks can affect the survivability of soldiers and their fellow troops, but there is nothing a soldier can do about it. After all, what are the legal consequences for allowing crackers around the world access to corporate data or a soldier to die on the battlefield? It is all about the bottom line in the corporate and government worlds. You and I have heard a lot about these massive breaches in their data servers resulting in stolen information, but little about the individuals being held accountable for this negligence or incompetence. At the very least, these breaches should result in some sort of change in management that might do better with the company's or government's cyber security requirements. For example, placing an experienced IT professional in charge rather that a crony or business graduate might make a difference. **The bottom line is that corporations should be punished for cutting costs at the expense of sacrificing their confidential data. We need accountability for these criminals that cost common citizens time and money to deal with the consequences!** |

I give these stories for a reason and that is because there is abundant evidence of how these practices rebound on corporations and their customers. I very much encourage you to read the Price Waterhouse Global Economic Crime Survey at <http://www.pwc.com/en_GX/gx/economic-crime-survey/assets/GECS_GLOBAL_REPORT.pdf>. Cybercrime now ranks as one of the top four economic crimes as businesses face serious threats from cyber criminals. The report points out those four in ten respondents to their survey say that their organizations do not have the capability to prevent or detect cybercrime. The report has many other disturbing statistics and facts to support the story above, so let's just cover a few:

* 1 in 10 who reported fraud suffered losses of more than U.S. $5 million.
* 56% of respondents said the more serious fraud was an 'insider job'.
* 2 in 5 respondents had not received any cyber security training.
* A quarter of respondents said that there is no regular formal review of cybercrime threats by the CEO and the Board.
* The majority of respondents do not have, or are not aware of having, a cyber-crisis response plan in place.

From the report, "*It’s not just about IT. It’s about HR making sure employees understand the security policies, and recruiting people with the specialist skills to protect the organization from cyber-attacks. It’s about legal and compliance making sure laws and regulations are respected. It’s about physical security protecting sites and IT equipment. It’s about marketing thinking about cyber security when they launch new products.*" The survey goes on to report, "*But our survey results suggest that the perception of cybercrime is changing, and that organizations are now recognizing the risk of cybercrime coming from inside. 53% of the respondents who said the cybercrime threat was an internal one believe that there is a risk from the information technology (‘IT’) department. It’s not surprising that many respondents think this, because they expect IT personnel to have the necessary skills and opportunity to commit these crimes. In particular, IT personnel might have ‘super user’ access, which gives them extra administrative rights to access systems and the ability to delete audit trails, making it harder to detect their wrongdoing.*"

In [Chapter 9](#_-_Hardening_security) we will cover SSH, where we will discuss the exchange of public/private key pairs for the exchange of data. However, in my experience I have seen how unknowledgeable IT managers direct their staff to do some crazy things. Throughout my many years in IT, I have written many scripts that automate the exchange of data between your home/business and the partners with whom you are working. GnuPG is a wonderful tool to use to encrypt and exchange data with your partners, but I doubt many of them know how to use it. I don't know of any schools or continuing education classes that you can send employees to that will teach them about this easy to use open source encryption technology. (See: [http://www.ibm.com/developerworks/aix/library/au-gnupg](http://www.ibm.com/developerworks/aix/library/au-gnupg/), <http://gnupg.org/gph/en/manual.html#AEN136>, <https://en.wikipedia.org/wiki/GNU_Privacy_Guard>)

### GnuPG -- Generating Key Pairs -- Verifying Pubic Keys for the Exchange of Encrypted Data and Email

In my latest Linux distribution, the version of GnuPG was at 1.4.16. However, at the open source website the latest version of this encryption software was at 1.4.18. You can check the version of GunPG by typing $ gpg --version at the command line to see what version your flavor of Linux is providing. You must do this before performing the following steps. To get the latest version or Linux GnuPG surf to <https://www.gnupg.org/download/index.en.html> and download the latest release of the software gunpg-1.4.18.tar.bz2 or gnupg-1.4.18.tar.gz When done downloading be sure to check the validity of the download by checking the SHA-1 checksum displayed on the web page by typing:

$ sha1sum gnupg-1.4.18.tar.gz

ea7d66c3de7aaf46de9e8678f4fc4a8c329400b2 gnupg-1.4.18.tar.gz

$ sha1sum gnupg-1.4.18.tar.bz2

41462d1a97f91abc16a0031b5deadc3095ce88ae gnupg-1.4.18.tar.bz2

Once downloaded you can unzip, configure, make and install the latest version by performing the following steps:

$ tar jxvf gnupg-1.4.18.tar.bz2

$ cd gnupg-1.4.18

$ su # change user to root

# cd gnupg-1.4.18

# ./configure

# make

# make install

So now you have multiple options of GPG to run on your Linux OS.

$ /usr/bin/gpg --version

gpg (GnuPG) 1.4.16

Copyright (C) 2013 Free Software Foundation, Inc.

$ /usr/local/bin/gpg --version

gpg (GnuPG) 1.4.18

Copyright (C) 2014 Free Software Foundation, Inc.

There is also a new GnuPG 2.0 modularized version of GnuPG supporting Open PGP and S/MIME, which their developers recommend. This will likely make three options available of GnuPG to run in you Linux OS. Installing this version becomes more complicated as there are also libraries that you need to first download and install. I found no other easy way to get this version of their software installed. The first step is to download the gnupg-2.0.x.tar.bz file from gnupg.org, expand the compressed file and view the README file.

$ sha1sum gnupg-2.0.25.tar.bz2

890d77d89f2d187382f95e83e386f2f7ba789436 gnupg-2.0.25.tar.bz2

$ tar jxvf gnupg-2.0.25.tar.bz2

$ cd gnupg-2.0.25 && vim README

In my case at the top of the README in the build instructions it stated that GnuPG 2.0 depends on the following packages:

libgpg-error (<ftp://ftp.gnupg.org/gcrypt/libgpg-error>)

libgcrypt (<ftp://ftp.gnupg.org/gcrypt/libgcrypt>)

libksba (<ftp://ftp.gnupg.org/gcrypt/libksba>)

libassuan >= 2.0 (<ftp://ftp.gnupg.org/gcrypt/libassuan>)

You also need the Pinentry package for most of the GnuPG functions; however it is not a build requirement. Pinentry is available at <ftp://ftp.gnupg.org/gcrypt/pinentry>. You should get the latest versions of course, the GnuPG configure script complains if a version is not sufficient. After building and installing the above packages in the order as given above, you may now continue with GnuPG installation (you may also just try to build GnuPG to see whether your already installed versions are sufficient). As 'root':

# tar jxvf libgpg-error-1.13.tar.bz2

# cd libgpg-error-1.13 && ./configure && make && make install && cd ..

# tar jxvf libgcrypt-1.6.1.tar.bz2

# cd libgcrypt-1.6.1 && ./configure && make && make install && cd ..

# tar jxvf libksba-1.3.0.tar.bz2

# cd libksba-1.3.0 && ./configure && make && make install && cd ..

# tar jxvf libassuan-2.1.1.tar.bz2

# cd libassuan-2.1.1 && ./configure && make && make install && cd ..

# tar jxvf pinentry-0.8.3.tar.bz2

# cd pinentry-0.8.3 && ./configure && cd ..

# tar jxvf gnupg-2.0.25.tar.bz2

# cd gnupg-2.0.25 && ./configure && make && make install && cd … && exit

# /usr/bin/gpg2 --version

gpg (GnuPG) 2.0.25

libgcrypt 1.6.1

Copyright (C) 2014 Free Software Foundation, Inc.

This will install a GnuPG binary in /usr/local/bin directory. So now you have three options of the software to choose to run. To get started using GnuPG we have to generate a key pair.

$ gpg --gen-key

gpg: directory `/home/testuser/.gnupg' created

gpg: new configuration file `/home/testuser/.gnupg/gpg.conf' created

gpg: WARNING: options in `/home/testuser/.gnupg/gpg.conf' are not yet active during this run

gpg: keyring `/home/testuser/.gnupg/secring.gpg' created

gpg: keyring `/home/testuser/.gnupg/pubring.gpg' created

Please select what kind of key you want:

(1) RSA and RSA (default)

(2) DSA and Elgamal

(3) DSA (sign only)

(4) RSA (sign only)

Your selection? 1

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048) 4096

Requested keysize is 4096 bits

Please specify how long the key should be valid.

0 = key does not expire

<n> = key expires in n days

<n>w = key expires in n weeks

<n>m = key expires in n months

<n>y = key expires in n years

Key is valid for? (0)

Key does not expire at all

Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID

from the Real Name, Comment and Email Address in this form:

"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: GPG User

Email address: gpguser@domain.com

Comment: GPG User ID for partner data exchange

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

Enter passphrase:

Repeat passphrase:

Whenever a key pair is generated it is important that the revocation certificate be generated as well. The certificate is needed in the event that you forget your passphrase or your private key is lost or compromised. A revocation certificate is published to notify your partners that your public key should no longer be used.

$ gpg --output gpguser\_revoke.asc --gen-revoke [gpguser@domain.com](mailto:gpguser@domain.com)

sec 4096R/11C12F2E 2013-08-14 GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

Create a revocation certificate for this key? (y/N) y

Please select the reason for the revocation:

0 = No reason specified

1 = Key has been compromised

2 = Key is superseded

3 = Key is no longer used

Q = Cancel

(Probably you want to select 1 here)

Your decision? 1

Enter an optional description; end it with an empty line:

> This gpguser key is no longer valid, remove it.

Since our private key is so very important, let's make a backup copy of it and store it in a very safe place.

$ gpg --export-secret-keys > > <userid>-privkey.gpg

$ gpg --export-secret-keys --armor <youremailaddress> > <userid>-privkey.asc

We now need to exchange keys with our partners; this means you have to export your public key to a file then send it to and verify it with your partners. The GnuPG tools support keys in binary and ASCII format. The binary format is appropriate if you can safely exchange files with your partner. If this is not possible, you can post an ASCII version of your public key on your website or send it in an email to your partner. The fingerprint verification previously discussed will ensure that they received the correct key.

$ gpg --list-keys

/home/testuser/.gnupg/pubring.gpg

---------------------------------

pub 4096R/11C12F2E 2013-08-14

uid GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

sub 4096R/90317E49 2013-08-14

$ gpg --output gpguser.gpg --export gpguser@domain.com

$ gpg --armor --output gpguser.asc --export gpguser@domain.com

To keep things organized you may want to maintain all of your GPG files in the keyring directory.

$ ls /home/gpguser/.gnupg

gpg.conf gpguser.gpg pubring.gpg random\_seed trustdb.gpg

gpguser.asc gpguser\_revoke.asc pubring.gpg~ secring.gpg

We are now ready to exchange keys with our partners and start adding their public keys to our keyring. When we receive or retrieve their public key, we need to import it to our public keyring.

$ gpg --import partner\_pub.gpg

gpg: key 66D4F240: public key "partner@domain.com (Test key) <partner@domain.com>" imported

gpg: Total number processed: 1

gpg: imported: 1 (RSA: 1)

$ gpg --list-keys

pub 4096R/11C12F2E 2013-08-14

uid GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

sub 4096R/90317E49 2013-08-14

pub 2048R/66D4F240 2013-08-08

uid partner@domain.com (Test key) <partner@domain.com>

sub 2048R/FD2E1B71 2013-08-08

Once the public key is added to our keyring we can view the fingerprint we need to verify with our partner.

$ gpg --fingerprint partner@domain.com

pub 2048R/66D4F240 2013-08-08

Key fingerprint = 2F5E 359E C718 0CAF 746B 366B 567D 7831 66D4 F240

uid kellis2@hotmail.com (Test key) <kellis2@hotmail.com>

sub 2048R/FD2E1B71 2013-08-08

### GunPG -- Signing and Assigning Trust Levels to Public Keys

Once the fingerprint has been verified to your level of satisfaction you should sign the key to show that it has been verified. When you sign a public key you are establishing a level of trust that you want to place on the key. If you are going to exchange the public key with others who trust you, be sure to *thoroughly vet the key*. Never sign a key where your partner has not provided their key length, key type, key id, creation date, expiration date if any, and their public key fingerprint. Your reputation and possible legal repercussions are on the line.

|  |  |
| --- | --- |
| **Story** | As an Electronic Warfare technician, I learned a lot about dealing with all sorts of devices that handled a lot of lethal amounts of voltages, amperes and all facets of the dangers faced in the field of electricity that your average home electrician would never encounter. Upon my return from the Iraq 2003 war, I decided to study home electrical codes and methods and found out that there was a lot I did not know about simple 120-volt systems. In the military, we study and work on complicated Electronic Warfare equipment that we have to deal with on the war front. So when we come back home, it is weird learning about the elementary electrical equipment civilians deal with on a day-to-day basis. It is kind of like watching the show "Are You Smarter Than a Fifth Grader" and find out sometimes you are not.  Using my years of experience, I quickly set about rewiring my house, building a new breaker box, and rewiring everything to all the latest electrical standards that I had studied. However, the one thing I was not secure with was hooking the whole arrangement up to the raw power lines coming into my house, so I asked coworkers for recommendations for someone to do this final step in my months-long project. Rather than pay a qualified electrician, I went with their recommendation and have had nightmares ever since. The electrician was a complete idiot, cut house wiring going into the new breaker box, which you should never do, reeked of alcohol, and so on. I was beside myself watching this nightmare take place, but what could I do? If I stopped the work, the best estimate I had gotten for the same work was about $600 more, and if I were them, I would charge me full price, as my house was without power on a hot summer day. I asked my coworkers how they could recommend this person and they said some friends said he was OK. **They had NO FIRST HAND KNOWLEDGE AS TO THE QUALITY OF HIS WORK!** Do you think I would ever trust anything they ever said again? |

The moral of the story above is never assign a level of trust to a partner's public key without properly verifying and vetting it. Below are the guidelines that I use to assign the trust levels on crypto keys:

1. No opinion means you know nothing about the key or won't assign a level of trust to it for your own reasons.
2. Do not trust means that you don't trust the key, but you have to exchange data with the creator for some reason.
3. Limited or marginal trust means that you verified the fingerprint and email address over the phone and verified the business address (that they exist at the address listed). This can be done with a signature required letter at the post office. Signature confirmation provides shippers with the name of the recipient as well as the delivery date, time, and location; a copy of the recipient's signature will also be faxed, mailed, or emailed to you upon request. This should be kept in a file cabinet for legal purposes. This can also mean that you trust someone else's verification of the key, just be sure you trust in the fact that they vetted the key thoroughly.
4. Casual or fully, means that, in addition to all of the above, verifying the user ID against a copy of a hard to forge photo ID of the key owner, such as a driver's license or preferably, a passport.
5. Extensive or ultimately, means all of the above, except verification is done in person with original hard to forge multiple documents such as a passport or birth certificate. Be extremely pedantic with checking the person's identity, as your reputation is on the line. Verifying keys correctly is what signing PGP keys is all about. You should also look up their business at the Better Business Bureau or some other service such as Angie's List [http://www.angieslist.com](http://www.angieslist.com/) to see if there are any problems reported with their business reported. The remaining task prior to signing is to determine whether the other party has control over their email. To check this, send them a file such as a random number encrypted with their key. The task of your partner is to decrypt this file or random number and send it back to you encrypted to your public key. It is very important that your partner sign the message or file they send back so that you can verify their identity. If they are technically incapable of this task give their key a lesser rating.

Of course there are exceptions that you will have to determine for yourself. For example, if you are dealing with a large corporation you might give them a 3 after just a phone call and an exchange of email if you feel confident with that information. It is unlikely that you will reach a computer professional at a corporation willing or capable of establishing a level 5 key rating. The management at the multi-billion dollar corporations that I worked for did not even inquire, know or care if I verified the keys or not. I do not believe they even knew that this was an important step in the process of exchanging encrypted data. To sign the key, you can edit the key or just sign it from the command line. However, editing the key will allow you to establish the level of trust that you want to place on the key, which is much more important.

$ gpg --sign-key 11C12F2E

$ gpg --edit-key [partner@domain.com](mailto:partner@domain.com)

gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

pub 2048R/66D4F240 created: 2013-08-08 expires: never usage: SC

trust: unknown validity: unknown

sub 2048R/FD2E1B71 created: 2013-08-08 expires: never usage: E

[ unknown] (1). partner@domain.com (Test key) <partner@domain.com>

gpg> fpr

This will also show the fingerprint of the key being verified.

gpg> sign

You need a passphrase to unlock the secret key for

user: "GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>"

4096-bit RSA key, ID 11C12F2E, created 2013-08-14

gpg> check

uid partner@domain.com (Test key) <partner@domain.com>

sig!3 66D4F240 2013-08-08 [self-signature]

sig! 11C12F2E 2013-08-14 GPG User (GPG User ID for partner data exchange

gpg> trust

pub 2048R/66D4F240 created: 2013-08-08 expires: never usage: SC

trust: unknown validity: unknown

sub 2048R/FD2E1B71 created: 2013-08-08 expires: never usage: E

[ unknown] (1). thatcybersecurityguy@outlook.com (Test key) <thatcybersecurityguy@outlook.com>

Please decide how far you trust this user to correctly verify other users' keys

(by looking at passports, checking fingerprints from different sources, etc.)

1 = I don't know or won't say

2 = I do NOT trust

3 = I trust marginally

4 = I trust fully

5 = I trust ultimately

m = back to the main menu

Your decision? 5

We can also view the trust status of a key from the command line.

$ gpg -kvv 66D4F240

gpg: using PGP trust model

gpg: checking the trustdb

gpg: 2 keys cached (5 signatures)

gpg: 2 keys processed (2 validity counts cleared)

gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model

gpg: depth: 0 valid: 2 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 2u

pub 2048R/66D4F240 2013-08-08

uid partner@domain.com (Test key) <partner@domain.com>

sig 3 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com.com>

sig 11C12F2E 2013-08-14 GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

sub 2048R/FD2E1B71 2013-08-08

sig 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com>

Now that you have signed your partner's public key, you can now provide that key to your other partners creating what GnuPG calls your "Web of Trust". The way this works is that, you have expended time and effort to verify and sign someone's public key. Someone else you know or do business with needs to exchange data with you and the partner that you verified. Therefore, what you can do is export that partners signed public key and send it to your third party, who can then choose to trust your endorsement of the key or not. You will also want to provide your copy of the signed key to your partner so they can add your signature to their keyring. Once your partner adds your signature to their public key, they can then export their public key with your signature to provide to people with whom they correspond. GnuPG suggests having "key signing parties" where everyone gets together to verify, sign, export and import everyone else's public keys. You can then all exchange data within your group with a high degree of certainty and security. (See: <https://en.wikipedia.org/wiki/Key_signing_party>, <http://www.cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html>,

<https://en.wikipedia.org/wiki/Linux_User_Group>)

To put another way, you have a partner that expends at lot of time and effort managing and verifying their OpenPGP keys. You have verified, vetted and assigned a level of trust to your partner's key. Your partner has thoroughly investigated and verified another person's public key at a key signing party that you are adding to your public keyring so that you can exchange encrypted email with them. The person who directly owns the public key has also fully trusted the key they are managing.

The above scenario creates a "Web of Trust" or a trust path from you to the public key of the person you want to exchange encrypted email with. You can now trust that public key without having to depend on certificate authorities, which cost money. Understand that this method of key trust is not perfect as it depends on a network of direct human relationships and it is hard to find qualified personal at the businesses you exchange data with that are capable of using OpenPGP.

$ gpg --list-sig

/home/partner/.gnupg/pubring.gpg

-------------------------------

pub 2048R/66D4F240 2013-08-08

uid partner@domain.com (Test key) <partner@domain.com>

sig 3 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com>

sub 2048R/FD2E1B71 2013-08-08

sig 66D4F240 2013-08-08 partner@domain.com (Test key) [partner@domain.com](mailto:partner@domain.com)

$ gpg --import /tmp/partner.gpg

gpg: key 66D4F240: "partner@domainl.com (Test key) <partner@domain.com>" 1 new signature

gpg: Total number processed: 1

gpg: new signatures: 1

gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model

gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u

$ gpg --list-sig

/home/partner/.gnupg/pubring.gpg

-------------------------------

pub 2048R/66D4F240 2013-08-08

uid partner@domain.com (Test key) <partner@domain.com>

sig 3 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com>

sig 11C12F2E 2013-08-14 [User ID not found]

sub 2048R/FD2E1B71 2013-08-08

sig 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com>

If you look above you see that User ID 11C12F2E has signed our public key. To see more information about whom they are import their public key.

$ gpg --import /tmp/gpguser.gpg

gpg: key 11C12F2E: public key "GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>" imported

gpg: Total number processed: 1

gpg: imported: 1 (RSA: 1)

$ gpg --list-sig

/home/partner/.gnupg/pubring.gpg

-------------------------------

pub 2048R/66D4F240 2013-08-08

uid partner@domain.com (Test key) <partner@domain.com>

sig 3 66D4F240 2013-08-08 partner@domain.com (Test key) <partner@domain.com>

sig 11C12F2E 2013-08-14 GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

Until you sign a key, GPG will question whether you want to use this added key to encrypt a file. Therefore, if you plan to automate processes through scripting or some other process, you should sign all of the public keys that you plan to automate using the public keyring. Below is an example of using an unsigned key:

$ gpg --encrypt --sign --yes --passphrase-file pass.txt -r 66D4F240 -r 11C12F2E <file>

Reading passphrase from file descriptor 3

You need a passphrase to unlock the secret key for

user: "gpguser@domain.com (Test key) <gpguser@domain.com>"

2048-bit RSA key, ID 66D4F240, created 2013-08-08

gpg: 90317E49: There is no assurance this key belongs to the named user

pub 4096R/90317E49 2013-08-14 GPG User (GPG User ID for partner data exchange) <gpguser@domain.com>

Primary key fingerprint: 14B8 675B 7518 E745 19B6 D628 E478 9919 11C1 2F2E

Subkey fingerprint: BF29 B994 CB06 0567 43D8 C00C 8931 60AD 9031 7E49

It is NOT certain that the key belongs to the person named

in the user ID. If you \*really\* know what you are doing,

you may answer the next question with yes.

Use this key anyway? (y/N) y

Even after using the quiet option, you may get the warning "gpg: WARNING: message was not integrity protected". You will receive this warning if the original file was not encrypted with modification detection checking (MDC) enabled. This warning can be suppressed by issuing the --no-mdc-warning when decrypting the file. However, it is best to encrypt the file with MDC in place. MDC will force the use of encryption with a modification detection code, which is always used with new ciphers that have a block size greater than 64 bits.

$ gpg --symmetric --force-mdc --passphrase "thisistosavethis" p.txt

$ gpg --symmetric --armor --force-mdc --passphrase "thisistosavethis" p.txt

To decrypt the file, use the following:

$ gpg --passphrase "thisistosavethis" --yes p.txt.gpg

gpg: CAST5 encrypted data

gpg: gpg-agent is not available in this session

gpg: encrypted with 1 passphrase

$ gpg --passphrase "thisistosavethis" --yes p.txt.asc

… same output as .gpg above.

As we discussed, there are levels of security for you to consider using encryption technologies such as GnuPG. The example above is one of encrypting a file to a simple passphrase. This is called symmetric key encryption because the same passphrase is used to both encrypt and decrypt a file. This is the simplest form of encryption that you can quickly employ to start exchanging encrypted data with your partners or protecting a few files on your computer. All that is needed is a working version of GnuPG and a known passphrase on one or both sides of the data exchange. In the example scripts in the next sections we will use this form of encryption to encrypt a file that houses our private key passphrase so that we can automate the decryption of data retrieved from our partners. We do this so that if a cracker or administrator gets access to our files, they would have to know which script we used to run to retrieve encrypted data and find the code in that script that encrypts and decrypts our passphrase file. This is not as easy as you might think, as corporations and governments may have many thousands of lines of scripting code running on their computers. No methodology is foolproof, but this method is better than having an unencrypted version of your passphrase lying around on your computer in easy to read text form. It is an added level of security that may keep a cracker from gaining access to your very private passphrase.

The GunPG web page [http://www.gnupg.org](http://www.gnupg.org/) also offers a less versatile, more sophisticated Windows GUI as a solution that we will discuss later. ***In the upper left click on Download > scroll down until you see GnuPG Binaries and next to Packages for MS-Windows are available at Gpg4win click on Gpg4win link.*** This will take you to [http://www.gpg4win.org](http://www.gpg4win.org/) where you can download their encryption software. There is also an Apple Gnu Privacy Guard encryption solution which can be found at <http://www.gpgtools.org>. You can use Gpg4win to validate a public key also, but this software does not offer us the option of assigning different levels of authentication. (See: <http://www.gpg4win.org/doc/en/gpg4win-compendium_16.html>)

Another method you can use is to download a keyring where the public key you obtained is already signed by the keys of others that have already trusted the key. For example, you can download the keys of all Debian developers by installing the debian-keyring package. You can then verify the signatures Debian developers have made with their keys on other public keys you need to use and verify. Much of the following tutorial was taken from the Tails web page <https://tails.boum.org/doc/get/trusting_tails_signing_key/index.en.html> which describes various ways to verify a key signature.

# apt-get install debian-keyring

To get a list of the signatures made by other people on the signing key you are verifying let's use the Tails signature and ISO file as an example:

$ gpg --keyid-format long --list-sigs <RSA key ID>

$ gpg --keyid-format long --list-sigs 1202821CBE2CD9C1

pub 4096R/1202821CBE2CD9C1 2010-10-07 [expires: 2012-10-06]

uid Tails developers (signing key) <tails@boum.org>

sig 3 1202821CBE2CD9C1 2011-04-16 Tails developers (signing key) <tails@boum.org>

sig BACE15D2A57498FF 2011-04-16 [User ID not found]

sig CCD2ED94D21739E9 2011-06-12 [User ID not found]

uid T(A)ILS developers (signing key) <amnesia@boum.org>

sig 3 1202821CBE2CD9C1 2010-10-07 Tails developers (signing key) <tails@boum.org>

sig BACE15D2A57498FF 2010-10-07 [User ID not found]

The lines ending in [User ID not found] mean that these are signatures from public keys that you have not verified and added to your keyring. To learn more about them you can search the Debian keyring using the 16 digit code between the "sig" tag and the date.

$ gpg --keyring=/usr/share/keyrings/debian-keyring.gpg --list-key CCD2ED94D21739E9

pub 4096R/CCD2ED94D21739E9 2007-06-02 [expires: 2012-05-31]

uid Daniel Kahn Gillmor <dkg@fifthhorseman.net>

uid Daniel Kahn Gillmor <dkg@openflows.com>

uid [jpeg image of size 3515]

uid Daniel Kahn Gillmor <dkg@debian.org>

sub 4096R/C61BD3EC21484CFF 2007-06-02 [expires: 2012-05-31]

sub 2048R/125868EA4BFA08E4 2008-06-19 [expires: 2011-05-31]

We can now import a key from the Debain keyring that trusts the Tails public key to our own keyring:

$ gpg --keyring=/usr/share/keyrings/debian-keyring.gpg --export CCD2ED94D21739E9 | gpg --import

After import, we can verify the signature made by this new key on the Tails signing key by typing:

$ gpg --keyid-format long --check-sigs 1202821CBE2CD9C1

pub 4096R/1202821CBE2CD9C1 2010-10-07 [expires: 2012-10-06]

uid Tails developers (signing key) <tails@boum.org>

sig!3 1202821CBE2CD9C1 2011-04-16 Tails developers (signing key) <tails@boum.org>

sig! CCD2ED94D21739E9 2011-06-12 Daniel Kahn Gillmor <dkg@fifthhorseman.net>

uid T(A)ILS developers (signing key) <amnesia@boum.org>

sig!3 1202821CBE2CD9C1 2010-10-07 Tails developers (signing key) <tails@boum.org>

sig! CCD2ED94D21739E9 2010-12-29 Daniel Kahn Gillmor <dkg@fifthhorseman.net>

pub 4096R/1202821CBE2CD9C1 2010-10-07 [expires: 2012-10-06]

uid T(A)ILS developers (signing key) <amnesia@boum.org>

sig!3 1202821CBE2CD9C1 2010-10-07 T(A)ILS developers (signing key) <amnesia@boum.org>

sig! CCD2ED94D21739E9 2010-12-29 Daniel Kahn Gillmor <dkg@fifthhorseman.net>

3 signatures not checked due to missing keys

From the Tails web page, "*On the output, the status of the verification is indicated by a flag directly following the "sig" tag. A "!" indicates that the signature has been successfully verified, a "-" denotes a bad signature and a "%" is used if an error occurred while checking the signature (e.g. a non supported algorithm).*" The above output shows that the signature of Daniel Kahn Gillmor on Tails signing key has been successfully verified.

If you are not comfortable using the command line to manage keys, there are graphical tools you can use in Linux that can help you. For example, in Fedora you can try "seahorse":

# yum install seahorse

You can then manage key by typing $ seahorse to bring up the Password and Key management tool. In the KDE Desktop we can use kgpg to create a GPG key pair.

(See: <https://fedoraproject.org/wiki/Cryptography>, <https://fedoraproject.org/wiki/Creating_GPG_Keys>, <https://help.ubuntu.com/community/GnuPrivacyGuardHowto>)

### GnuPG -- Digital Signatures and Why They Are Important

Wiki defines a digital signature as "*a mathematical scheme for demonstrating the authenticity of a digital message or document.*" A digital signature is an added measure of security that allows the recipient of a message or document to verify its integrity. By integrity, we mean it shows that the data was not altered in transit so that we know it was not tampered with or forged. Some schemes also associate a time stamp with when the data was signed thus showing the signature is valid. The signature is used to authenticate the sender of the document or message and to verify that it was not changed during transit.

It shows the recipient that the message or document was created by a known sender much like how we view a hand written signature. While "gpg" can be used to encrypt and decrypt files to send to your partners, it can also be used to sign those documents. Encryption can keep someone from understanding a message or deciphering a document; it does not prevent someone from altering the encrypted data. A signature allows the receiver to verify that the file or message has not been altered since the signature was assigned. For example, a cracker could apply a fake signature to a document encrypted to your public key, thus impersonating you, or perhaps a financial partner has to know that you cannot later deny having signed the transaction that you asked them to conduct, thus having legal significance. A digital signature can be applied to both encrypted and unencrypted messages to show that they have arrived intact and unaltered. Here is how this works:

1. You create and email a correspondence or document that your receiving party needs to know is unaltered upon receipt.
2. You use software such as GnuPG to obtain a hash, checksum, Cyclic Redundancy Check or some other means to "hash" the message thus creating a "message digest" so that upon receipt it can be proven to be authentic.
3. That same software uses your private key and your partner's public key to encrypt the "message digest" and appends or surrounds the document or message. This encrypted digest becomes your signature and is different for every message you send. Since this message digest is encrypted to your private key, which only you possess, it becomes your digital signature. Hopefully this shows how important protecting your digital private key is!
4. The GnuPG software then appends the encrypted message digest to the document or message, thus signing it.

Below is what a signed message looks like before being decrypted:

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

What happens when I sign this...

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v2.0.22 (MingW32)

Comment: Using GnuPG with Thunderbird - http://www.enigmail.net/

iQEcBAEBAgAGBQJSzNffAAoJEFU80nvnBjsjITIH/36o3Fjj4XHzOaodEIeciTvY

ppT3LEq2KsnwOuz/e0HtVixyJ4dvKRdzEuTH/8S3NiQ4EkzcEDB2EbXhOykc6XjZ

WlcRXIrRM+hFbIWhniVj3oaViFMm1WchUlZ5rxotfuWvhl77NtBZWIGHhuzSTn4y

LEBzewxN03kTP4rrPJUaSNGfmbngMvgzXP4eC0Lgf0mtBjQnXArgEIB8+0KXlHE1

wph5yBxlrXBsHilbyPdtB7efr4vvbSADmVhw5RmaaDLv5jqAmOPh49/7bdzYmQtk

UhwSt4o4WHZml5UrDoVHHD28tcUx3VKb+jjq8pg8OljYenNub0bzpyy/boNz4pU=

=XFC5

-----END PGP SIGNATURE-----

When your partner receives your correspondence:

1. When your partner receives the data, they use their software to decrypt the signature using your public key and their private key, thus changing it back into a message digest. If your partner can decrypt your signature, this proves you sent the message since it was encrypted to your private key, which only you possess.
2. The GunPG software then makes a hash of your decrypted document or message to compare to the message digest that was attached as your signature, and if they match, your partner knows that the data has not been changed during transit.
3. The decryption software will tell your partner that this message contains a good signature from you, thus verifying the message or document.

This sounds somewhat complicated, but the software does everything for you. (See: <https://en.wikipedia.org/wiki/Digital_signature>)

### GnuPG -- Scripting Encryption and File Transfer with Your Partners

Now that we have everything set up to exchange encrypted data with our partners, we have to create a low-cost infrastructure to do so efficiently and automatically. While we could assign an employee to perform the repetitive steps necessary to encrypt, decrypt, sign, send and receive encrypted data, this could become an ever-expanding process that would prove very painful and costly to have someone do day after day in your home or business. Odds are you don't have the resources to have someone who works for you trained and dedicated to this task. The answer for most SBOs as they conduct their core business is to pay an expensive service to handle these tasks for them, which is essentially the same thing as having an employee dedicated to this task.

Therefore, rather than present step-by-step directions and have you figure out how to incorporate encryption and file transfer to/from your SB/HC, I chose to show you some scripting that you can use to automate everything to exchange encrypted data with your partners. Odds are you know an experienced computer professional in your social sphere, or can temporarily employ a local college student who can use my example below to quickly adapt it to your business's needs. If the script below appears above your pay grade, it is not to individuals in the computer profession. You can hire an experienced Linux professional wanting some extra money to come into your small business and adapt the code below to exchange encrypted data with your business partners. While disabled, I dreamed of someone asking me to do this type of work.

I spent many years of my life scripting and automating encrypted file exchange to and from multi-billion dollar corporations with various other corporate and government partners. I could have written a book on this one topic alone, but I doubt many HCU/SBO users would have found a book dedicated to that topic very useful. I would have enjoyed presenting my technical work, constantly coding all of this, but that would have taken many pages and would be beyond the abilities of someone not skilled in writing scripts or computer software in all the various computer languages that I have worked in during my career. In addition, that book would have only landed me a job working for the government or corporations that I'm trying to help you understand are spying on everything you are doing in your Internet life. If I presented the limited information that highly paid corporate and government employees know, it would have defeated the purpose of this book.

Therefore, I chose to present the sample Linux BASH script below in the hopes that you can adapt it to your needs. Some of the original scripts and programs I wanted to present were over 5,000 to 14,000 lines in length. So, while the simple script below may seem overwhelming, it is a fraction of what I had running in the background at various corporations and government agencies. I tried to limit the script below to what I felt would benefit you the most. Perhaps, if this book is successful and you want further examples of coding, you can contact me via my website <http://thatcybersecurityguy.com> or my blog <http://thatcybersecurityguy.blogspot.com>, and I will get some scripts up or include them in my next revision of this book.

Before we get into the complexity of the script below, there are a few things that we want to set up for data exchange with our partner. In [Chapter 9](#_-_Hardening_security) we will discuss setting up an SSH server and the creation and use of partner SSH user IDs for the encrypted exchange of data. Once those IDs are created by both parties you have to mutually agree as to how the encrypted exchange of data will take place. Below are a few tips that I have used in the past to automate the exchange of data. Underneath the $HOME directory in Linux create the following directory tree:

$ mkdir -p <companyname>/in && mkdir <companyname>/out

$ mkdir <companyname>/in/archive && mkdir <companyname>/out/archive

The "in" directory is where you both deposit files that you each need to process and the out directory is from where you both retrieve files. To automate things between you and your partner's SSH servers you will need to script getting a directory listing of the *out* directory on your partner's server to see if there are files to retrieve. If there are files there you can retrieve them to your SSH server and process them. Once the files are decrypted and processed, you probably want to archive them for possible reference later. There are many strategies that can be used to archive files, which are also covered in [Chapter 11](#_-_Hardening_security). The script below will use variables such as 'UserID', 'CompanyName', etc. as the script needs to be able to support multiple partners.

The SSH scripting in [Chapter 9](#_-_Hardening_security) will duplicate some of this code and I will show you a much more sophisticated requirements definitions about the automated exchange of data. In my attempt to present my many years of work on projects such as this, I had to break it and thousands of lines of scripting in these technologies down to be able to show you a few simple examples in this book that I hope will provide the most benefit to you. In defining these SBO/HCU requirements, I hope I am presenting an example you can expand upon and adapt to your purposes.

|  |
| --- |
| **Requirements Definition for Automated Encrypted File Transfer and Decryption** |
| After the manual setup of the infrastructure necessary to exchange encrypted files with our various partners, this script shall automate various processes for the exchange of that data.   1. The script shall scan for, analyze, retrieve, decrypt and archive any new 'in' data files that our partner places in their remote SSH server 'out' directory. 2. This script shall scan for, analyze, encrypt, send, and archive files placed in our SSH server partner's account 'out' directory to our partner's SSH servers 'in' directory. 3. The script shall provide the capability to transfer unencrypted files to/from both SSH servers. 4. The script shall provide the capability to archive and encrypt files placed in our SSH server's 'in' directory. |

While this requirements definition sounds very simple, it is actually **very broad in scope**. This is an example of what IT professionals may face if an unknowledgeable manager without an IT background is placed in charge of an IT project. Some managers do not understand the value a well-thought-out *narrow* "Requirements Definition" is and how it can prevent scope creep during an IT development project. Their careers are on an entirely different path, and their IT staffs have to step up and demand limiting the projects to a tapered set of goals. However, this is akin to asking a business major to build a bridge or design a nuclear power plant. In most businesses this reflection does not happen often, and IT scope creep is often allowed to occur from a poor "Requirements Definition". We talked about whistleblowers previously, and from personal experience, if IT professionals object too loudly to the scope creep and course misdirection of an IT project, they are quickly ostracized as troublemakers and subject to removal. This occurs in all professions, but much more often during IT projects, to IT professionals. This science is very much misunderstood by most businesses that are not making a profit off their IT workers.

As stated above, the requirements definition above may sound simple but the scripting or coding to satisfy the needs of this project could become very **broad** in scope and could get very complex depending on how our requirements change. Unless things are narrowed down, this project could suffer some very serious scope creep. Scope creep is the allowance of uncontrolled changes or continuous growth in a project's scope that the unknowledgeable patrons holding the purse strings allow. Government contractors and corporations use this methodology to suck in billions of taxpayer dollars because sometimes the people running these projects do not understand the technology that they are managing. They are sometimes placed in charge of these projects as a political reward, crony favoritism or are just misadvised by the technical IT professionals that they are relying upon.

Take the example above and understand archiving alone could require months of work and thousands of lines of code depending on the government or business requirements and evolving requirements. For example, the statement "and archive files" could take on a life of its own resulting in thousands/million/billions of dollars. Archiving legal or clandestine information requirements that may be needed to house this data for years versus days or months imply the need for expensive backup solutions. The U.S. government needs the NSA's two billion dollar archive facility to record all Internet activity conducted by U.S. citizens. That data needs to be readily accessible for their sophisticated algorithms to peruse. For your SBO, there may be special naming requirements for archive files, such as the need to append a prefix or suffix to the archived files, attach a version number to the name of the file, change the owner or group the archive files belong to, attach a payroll run number to the file, compress the files, attach a date to the files, or in the case presented below, just rotate the archived files a defined number of times.

As we deal with partners, they will have their own processes setup. Some will post files on their server for us to process that coexist with files that we have already processed. Consequently, if we are automating things, we may have to script code to determine which files have been processed and which have not. Some partners will delete the files once they know we have successfully downloaded them. Some partners will want us to manage the files on their server, while others will not let us delete or move anything on their server. Some applications cannot distinguish between processed data and duplicate data. These are not simple tasks and in our business, coding to automate these tasks can vary from partner to partner, and application to application.

The script below is meant to be an example that satisfies the requirements above as simply as possible. I hope you can see how incredibly complex the automation of the exchange of encrypted data can become as each entity (corporation/government) we deal with may have differing requirements. These complications are why HCU/SBOs pay large sums of money to companies to handle these services for them. However, with this book I'm hoping to show how this type of automation is not out of the SBO/HCU's reach and give you some possible, in-house low-cost solutions. Doing these things on your own can save your business enormous sums of money, but I understand the need to concentrate on your core business and pay someone to perform these services for you. This works if you are vastly profitable, but perhaps you are like me, writing this book on a $22,000 a year budget. You may want to hire someone unemployed and skilled in Linux scripting to handle these tasks for you. God knows I would have taken this work and run with it while writing this book, but there were no offers. Once this initial infrastructure is in place, your systems will require minimal tweaks as requirements change or as you have to deal with new partners whom you can hire out to temporary workers or your local unemployed IT professional. By studying this book you now know what to look for in paid services, and that should help you find the best workers to fit your needs.

#!/bin/bash

# SCCS File %P%

# %Z% %M%: %Q% Version %I%.

# Made on %G% at %U%.

#

# Script is best viewed with "set ts=4".

#

set -x

#==========================================================================

#

# {Description}

# See Requirements Definition documentation.

#

# {Usage}

# See usage procedure below.

#

# {Description}

# The script uses Hungarian Notation prefixes as developed by Microsoft:

#

# strVar - string

# bVar - boolean

# iVar - integer

#

# GPG Notes:

#

# gpg --gen-key -> Generate a new key pair.

#

# Generate a key revocation certificate for your public key in case

# your private key is ever lost or compromised:

# gpg --output <mykeyrevoke.asc> --gen-revoke mykey

#

# gpg --list-public-keys # List all keys from the public keyrings.

# gpg --list-secret-keys # List all keys from the secret keyrings.

# gpg --import <partnerpubkey> # Import partners public key.

# gpg --delete-key <partnerpubkey> # Delete partners public key.

#

# How to encrypt files to public keys:

# gpg -e -r <YourPubKeyID> -r <CoPubKeyID> <file>

# gpg --encrypt --recipient <YourPubKeyID> --recipient <CoPubKeyID> <file>

#

# Create ASCII armored output:

# gpg -e -r <YourPubKeyID> -r <CoPubKeyID> --armor <file>

#

# Encrypt to public keys and write to 'outputfile.gpg':

# gpg -o <outputfile.gpg> -e -r <YourPubKeyID> -r <CoPubKeyID> <file>

#

# Decrypt the file given on the command line:

# gpg --decrypt <file>

#

# List all keys on the keyring and their fingerprints:

# gpg -kvc

# gpg --fingerprint

# gpg --fingerprint <CoPubKeyID> -> List the keys for the specified company

#

# gpg --output somepubkey.gpg --export name@domain.com # Export public key

# in binary format for name@domain.com.

# gpg --armor --export name@domain.com # Export public key in ASCII format

# for name@domain.com

# gpg --import somepubkey.gpg # Import somepubkey.gpg that someone sent to

# your public keyring.

# gpg --symmetric <text file> # To encrypt to a passphrase.

# gpg --symmetric <text file> --passphrase <passphrase> -> To encrypt and

# specify the passphrase on the command line.

# gpg --symmetric <text file> --passphrase-file <passphrasefile> # To

# encrypt and specify the passphrase file on the command line.

# gpg --passphrase-file <file> --yes <file>.gpg # Decrypt symmetric file

# with password file <file>.

#

# gpg --edit-key name@domain.com # Command> fpr -> To get fingerprint to

# verify by phone with customer.

# Command> sign -> To sign their public key to validate it.

#

# To view the signature for keys on your keyring:

# gpg -kvv

# gpg --list-sig

# gpg --list-sig <partnerpubkey>

# gpg --verify <file> # Verify a signed file.

#

# gpg --output file.gpg --encrypt --recipient someone@domain.com,me@domain.com file

# Encrypts a file to an intended recipient and my public keys.

# gpg --output file --decrypt file.gpg # Decrypt file.gpg to your private key.

# gpg --armor --export alice@cyb.org # Exporting a public key to text file.

#

# $HOME/.gnupg/secring.skr --> contains the private portion of the key pair.

# $HOME/.gnupg/pubring.pkr --> contains the public key.

#

# Script static changing variables:

typeset -x strCryptedFiles # Name(s) of the encrypted or decrypted files

#==========================================================================

# Declare static CONSTANT variables. They are declared here in the

# header for ease of change.

#

BREAK="-----------------------------------------------------------------------------"

typeset -x strBaseName="`basename ${0}`" # Get name of running script

typeset -x strOurGPGKeyID=0xFD2E1B71

typeset -x strPassDir=$HOME/.pass

#

#==========================================================================

#

# Declare boolean and character command line static variables:

# Declare static company specific boolean and string variables (set in

# procedure procSetCompanyVars) or passed in via the command line:

#

typeset -x strCompany # Command line company specified to be

# processed on our server

typeset -x strGPGFormat # GPG Format to be used

typeset -x strUser # Connection User ID

typeset -x strPass # Connection Password

typeset -x strSubDir # The subdirectory to process on our

# partners server

typeset -x strWorkingDir # Directory we are getting or putting

# to/from on our company SSH server

typeset -x strCoGPGKeyID # Company GPG Public Key ID

typeset -x strFilesToProcess # List of file(s) being processed

typeset -x strSSHAddress # SSH IP Address of remote host

typeset -x strLogsDir # Script log file directory

typeset -x strLogFile # Log of script actions

typeset -x strLSofDestDirFile # On get we compare previous "ls" of incoming

typeset -x strTrajectory=get # -t command line argument defaulted to

# to 'get' which means don't get or put

# the files being process to SSH server.

typeset -x strEmailList # Application owners to be notified on our

# local network or server

typeset -x bDebug=/bin/false # -d command line argument

typeset -x bEnDeCrypt=/bin/false # -e command line argument

typeset -x bTestMode=/bin/true # -s command line argument

typeset -x bPutGet=/bin/false # -p command line argument

typeset -x bFileSpecified=/bin/false # -f command line argument

#==========================================================================

# Send an Email with subject $1 and Email addresses $2. The current log

# file is concatenated into the Email.

#==========================================================================

procSendEmail()

{

if ${bDebug}; then

set -x

echo "Entering procSendEmail" >> ${strLogFile}

fi

strSub=$1

strMList=$2

strTmpMailFile=/var/tmp/mailfile.tmp

strHost=`hostname`

date +"%d%b%y(date +%H)$(date +%M)" > ${strTmpMailFile}

cat ${strLogFile} >> ${strTmpMailFile}

mailx -s "${strHost}-${strSub}" ${strMList} < ${strLogFile}

rm -f ${strTmpMailFile}

}

#==========================================================================

# Severity codes in $1 are as follows:

# - 1 means log a warning and continue processing

# - 2 means log a Fatal Error and exit 1

# - 3 means log a Fatal Error, send an Email and exit 2

#==========================================================================

procLogAbort()

{

# set -x

typeset -i iSeverity=$1

strMsg="$2"

if [[ ${iSeverity} -eq 3 ]]

then

printf "Fatal Error: ${strMsg}, aborting...\n" | tee -a ${strLogFile}

printf "Finished processing of Company ${strCompany} on `date`\n" | tee -a ${strLogFile}

procSendEmail "Fatal Error: ${strMsg}" ${strEmailList}

exit 1

elif [[ ${iSeverity} -eq 2 ]]

then

printf "Fatal Error: ${strMsg}, aborting...\n" | tee -a ${strLogFile}

printf "Finished processing of Company ${strCompany} on `date`\n" | tee -a ${strLogFile}

exit 1

else

printf "Warning: ${strMsg}\n" | tee -a ${strLogFile}

fi

}

#

# Set the path to the GPG command. Some flavors of Linux do not have

# the latest gpg binary in their repositories.

# NOTE: Must come after procedure procLogAbort.

#GPGCMD=`which gpg` # Yields /usr/bin/gpg at version 1.4.11

GPGCMD=/usr/local/bin/gpg # Downloaded from http://gnupg.org at version

# 1.4.14

if [ Y"$GPGCMD" = Y ]

then

procLogAbort 3 "'gpg' not found, variables hard coded..."

fi

#==========================================================================

# Display the list of partner companies this script supports.

#==========================================================================

procDisplayCompanyHelp()

{

echo "Valid companies are:"

echo

echo " lifeinsurance"

echo " legalservices"

echo " benefitservices"

echo " payrollservices"

echo " localsshserver"

echo " testcompany"

}

#==========================================================================

# Display the supported GPG crypt formats.

#==========================================================================

procDisplayGPGHelp()

{

echo "Valid GPG formats are:"

echo

echo " ASCII"

echo " ASCIISigned"

echo " Binary"

echo " BinarySigned"

}

#==========================================================================

# Display script usage help.

#==========================================================================

Usage()

{

echo "Usage:"

echo " $0 -c <company name> Specify company to process, required!"

procDisplayCompanyHelp

echo

echo " -a <GPG Format> Specify the GPG format of the input or output File"

procDisplayGPGHelp

echo

echo " -d Enable debug logging."

echo " -e Enable file encryption or decryption."

echo " -f <filename> Override default file to process with this file."

echo " -h Display usage."

echo " -p Enable SSH file transfer with partner."

echo " -s Run in Production mode. Default is Test."

echo " -t <trajectory> Specify SSH get or put, default is get."

}

#==========================================================================

# Get and process the command line arguments.

#==========================================================================

while getopts a:c:def:hpst: option

do

case $option in

a)

strGPGFormat=$OPTARG

echo "GPG Format set to ${strGPGFormat}..."

;;

c)

strCompany=$OPTARG

echo "Company to process is ${strCompany}..."

;;

d)

bDebug=/bin/true

echo "Debug logging enabled..."

;;

e)

bEnDeCrypt=/bin/true

echo "Encrypt or Decrypt files enabled..."

;;

f)

bFileSpecified=/bin/true

strFilesToProcess=$OPTARG

echo "File to process is ${strFilesToProcess}..."

;;

h)

Usage

exit

;;

p)

bPutGet=/bin/true

echo "Send or receive files from partner enabled..."

;;

s)

bTestMode=/bin/false

echo "Run in PROD mode enabled..."

;;

t)

strTrajectory=$OPTARG

if [[ ${strTrajectory} != "get" && ${strTrajectory} != "put" ]]

then

procLogAbort 2 "Invalid trajectory ${strTrajectory} specified"

fi

echo "Trajectory for file transfer is ${strTrajectory}..."

;;

\*)

Usage

procLogAbort 2 "Invalid command line argument '$option' specified..."

;;

esac

done

if [[ ! bEnDeCrypt && ! bPutGet ]]; then

Usage

procLogAbort 2 "Script not told to encrypt/decrypt files, or to retrieve/send files..."

fi

#------------------------------------------------------------------------------

# Log variable values for company being processed. This information is needed

# to determine how the script was run and what data was processed.

#------------------------------------------------------------------------------

procLogVars()

{

if ${bDebug}; then

set -x

echo "Entering procLogVars()"

fi

echo "Company being processed-------> ${strCompany}" >> ${strLogFile}

if ! ${bFileSpecified}

then

echo "File being processed----------> ${strFilesToProcess}" >> ${strLogFile}

fi

echo "Environment is----------------> ${strEnvironment}" >> ${strLogFile}

echo "Company GPG Public Key ID-----> ${strCoGPGKeyID}" >> ${strLogFile}

echo "Encrypt or Decrypt Format-----> ${strGPGFormat}" >> ${strLogFile}

if [[ -n ${strSSHAddress} ]]; then

echo "Company SSH Address-----------> ${strSSHAddress}" >> ${strLogFile}

echo "Company SSH User Name---------> ${strUser}" >> ${strLogFile}

echo "Company SSH Password----------> ${strPass}" >> ${strLogFile}

echo "Performing SSH----------------> ${strTrajectory}" >> ${strLogFile}

fi

if [[ ${strTrajectory} = "get" ]]; then

echo "Inbound DIR for SSH get-------> ${strWorkingDir}" >> ${strLogFile}

elif [[ ${strTrajectory} = "put" ]]; then

echo "Outbound DIR for SSH put------> ${strWorkingDir}" >> ${strLogFile}

else

echo "File will be en/decrypted only> ${strWorkingDir}" >> $strLogFile}

fi

echo "${strBaseName} Logging DIR--------> ${strLogsDir}" >> ${strLogFile}

echo "${strBaseName} Logging To---------> ${strLogFile}" >> ${strLogFile}

if [[ -n ${strLSofDestDirFile} ]]; then

echo "${strBaseName} LS of Dest Log-> ${strLSofDestDirFile}" >> ${strLogFile}

fi

if [[ -n ${strSubDir} ]]; then

echo "Company Subdirectory----------> ${strSubDir}" >> ${strLogFile}

fi

if [[ -n ${strEmailList} ]]; then

echo "Error Notification list-------> ${strEmailList}" >> ${strLogFile}

fi

}

#------------------------------------------------------------------------------

# If the file exists then

# - Move filename.2 to filename.3 and so on...

# - Move filename.1 to filename.2

# - Move filename.0 to filename.1

# - Move filename to filename.0

#------------------------------------------------------------------------------

function procStash

{

if ${bDebug}; then

# set -x

echo "Entering stash"

fi

#

# Sometimes routine is called with file names that don't exist. If the

# file does not exist we ignore the request to stash.

if [[ ! -f ${1} ]]

then

return 1

fi

typeset -i l k=20

while [ $k -gt 0 ];

do

l=$k-1

if [ -f ${1}.${l} ]; then

mv -f ${1}.${l} ${1}.${k} 1>>${strLogFile} 2>&1

fi

# if [ ! -s ${1}.${k} ]; then

# rm ${1}.${k} # Remove zero byte files

# fi

k=$k-1

done

if [ -f ${1} ]; then

mv -f ${1} ${1}.0 1>>${strLogFile} 2>&1

chmod 660 ${1}.0 1>>${strLogFile} 2>&1

fi

return 0

}

#------------------------------------------------------------------------------

# On all SSH puts we check for the existence of the file(s) being processed.

# If the file specified is not a file abort, if it is empty log a warning

# message.

#------------------------------------------------------------------------------

procCheckFiles()

{

if ${bDebug}; then

set -x

echo "Entering procCheckFiles()" >> ${strLogFile}

fi

typeset fFile=$1

cd ${strWorkingDir}

if [[ -z ${fFile} ]]; then

fFile=${strFilesToProcess}

fi

for i in ${fFile}

do

if [[ ! -f ${i} ]]; then

procLogAbort 3 "File ${strWorkingDir}/${i} not found."

fi

if [[ -z ${i} ]]; then

procLogAbort 1 "File ${strWorkingDir}/${i} is empty."

fi

done

}

#------------------------------------------------------------------------------

# On all SSH puts we check for the existence of the file(s) being processed.

#------------------------------------------------------------------------------

procCheckForOneFile()

{

if ${bDebug}; then

set -x

echo "Entering procCheckForOneFile()" >> ${strLogFile}

fi

typeset strFileDesc=$1

typeset fFile=$2

typeset -i iCnt=0

if [[ -z ${fFile} ]]

then

printf "No file found at ${strWorkingDir}" | tee -a ${strLogFile}

printf "containing string ${strFileDesc}" | tee -a ${strLogFile}

procLogAbort 3 "Checking for NULL file string is not allowed"

fi

cd ${strWorkingDir}

for i in ${fFile}

do

if [[ ! -f ${i} ]]

then

procLogAbort 3 "File ${strWorkingDir}/${i} not found"

fi

iCnt=$(( ${iCnt} + 1 ))

done

if [ ${iCnt} -gt 1 ]

then

printf "There is supposed to be only one file." | tee -a ${strLogFile}

procLogAbort 3 "More that one ${fFile} found"

fi

}

#------------------------------------------------------------------------------

# Move file specified in parameter 1 to the destination specified in

# parameter 2.

#------------------------------------------------------------------------------

procMoveFile()

{

if ${bDebug}; then

set -x

echo "Entering procMoveFile()" >> ${strLogFile}

fi

typeset strFromFile=$1

typeset strToFile=$2

strBreak="\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

if [ ! -f ${strFromFile} ]

then

echo "Attempt to move a file that does not exist!" >> ${strLogFile}

procLogAbort 3 "File `pwd`/${strFromFile} not found"

fi

mv -f ${strFromFile} ${strToFile} 1>>${strLogFile} 2>&1

if [[ $? -ne 0 ]] # Check the return status

then

echo ${strBreak} >> ${strLogFile}

echo "ll listing of directory `pwd`" >> ${strLogFile}

ls -al >> ${strLogFile}

echo ${strBreak} >> ${strLogFile}

procLogAbort 3 "'mv -f ${strFromFile} ${strToFile}' failed with status $?."

fi

}

#------------------------------------------------------------------------------

# Remove all files named in the static variable "strFilesToProcess".

#------------------------------------------------------------------------------

procRemoveFiles()

{

if ${bDebug}; then

set -x

printf "Entering procRemoveFiles()" >> ${strLogFile}

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

if [[ "" = "${strFilesToProcess}" ]]

then

procLogAbort 2 "No Files found to remove where files were expected..."

else

for i in ${strFilesToProcess}

do

rm -f ${i} 1>>${strLogFile} 2>&1

done

fi

}

#------------------------------------------------------------------------------

#------------------------------------------------------------------------------

procArchiveFiles()

{

if ${bDebug}; then

set -x

echo "Entering procArchiveFiles()" >> ${strLogFile}

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

if [ ! -d archive ]; then

echo "No archive directory found, executing 'mkdir archive'" >> ${strLogFile}

mkdir archive 1>>${strLogFile} 2>&1

fi

for j in ${strFilesToProcess}

do

if [[ ! -f ${j} ]]; then

procLogAbort 3 "Archive attempt on ${j} failed, file does not exist"

fi

echo "Archiving `ls -lF ${j}`" >> ${strLogFile}

procMoveFile ${j} archive

done

}

#------------------------------------------------------------------------------

# This function encrypts to the file(s) specified in the variable

# 'strFilesToProcess' to ours and our partners' public key in ASCII armor

# or BINARY format.

#------------------------------------------------------------------------------

procEncryptFiles()

{

if ${bDebug}; then

set -x

echo "Entering procEncryptFiles()" >> ${strLogFile}

fi

typeset strTmp=""

typeset -i iCnt=0

if [[ ${strGPGFormat} = "ASCII" ]]

then

strExt=".asc"

strOption="--armor"

else # The default is BINARY

strExt=".gpg"

strOption=""

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

for j in ${strFilesToProcess}

do

if [ -f ${j}${strExt} ]

then

echo "${j}${strExt} already exists, removing..."

rm -f ${j}${strExt}

fi

echo Encrypting: `ls -lF ${j}` >> ${strLogFile}

# gpg -e -r ${strCoGPGKeyID} -r ${strOurGPGKeyID} ${strOption} ${j} 1>>${strLogFile} 2>&1

echo "gpg -e -r ${strOurGPGKeyID} ${strOption} ${j} 1>>${strLogFile} 2>&1"

gpg -e -r ${strOurGPGKeyID} ${strOption} ${j} 1>>${strLogFile} 2>&1

if [[ ! -f ${j}${strExt} ]]

then

procLogAbort 3 "Unable to create ${strGPGFormat} GPG file ${j}${strExt}"

fi

echo Encryption file: `ls -lF ${j}${strExt}` >> ${strLogFile}

iCnt=$(( ${iCnt} + 1 ))

chmod 660 ${j}${strExt} 1>>${strLogFile} 2>&1

strTmp="${j}${strExt} ${strTmp}"

done

if [[ iCnt -eq 1 ]]; then

strCryptedFiles=${strFilesToProcess}${strExt}

else

strCryptedFiles=${strTmp}

fi

}

#------------------------------------------------------------------------------

# pgp -es <textfile> <Recipient KeyID> -u <your\_userID> -> to sign a

# plaintext file with your secret key and encrypt it with the

# recipient's public key in binary format.

#------------------------------------------------------------------------------

procEncryptAndSign()

{

if ${bDebug}; then

set -x

echo "Entering procEncryptAndSign()" >> ${strLogFile}

fi

typeset strTmp=""

typeset -i iCnt=0

if [[ ! -f ${strPassDir}/p.txt.gpg ]]; then

procLogAbort 3 "Pass phrase file ${strPassDir}/p.txt.gpg not found"

fi

cd ${strPassDir} 1>>${strLogFile} 2>&1

${GPGCMD} --passphrase "thisistosavethis" --yes p.txt.gpg 1>>${strLogFile} 2>&1

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

for j in ${strFilesToProcess}

do

if [ -f ${j}.gpg ]

then

echo "${j}.gpg already exists, removing..."

rm -f ${j}.gpg

fi

echo Encrypting: `ls -lF ${j}` >> ${strLogFile}

${GPGCMD} --encrypt --sign --yes --passphrase-file ${strPassDir}/p.txt \

-r ${strCoGPGKeyID} -r ${strYourGPGKeyID} ${j} 1>>${strLogFile} 2>&1

if [[ ! -f ${j}.gpg ]]

then

cd ${strPassDir} 1>>${strLogFile} 2>&1

${GPGCMD} --yes --symmetric --passphrase "thisistosavethis" p.txt 1>>${strLogFile} 2>&1

rm -f p.txt

procLogAbort 3 "Unable to create GPG file ${j}.gpg"

fi

echo Encryption file: `ls -lF ${j}.gpg` >> ${strLogFile}

iCnt=$(( ${iCnt} + 1 ))

chmod 660 ${j}.gpg 1>>${strLogFile} 2>&1

strTmp="${j}.gpg $strTmp"

done

cd ${strPassDir} 1>>${strLogFile} 2>&1

${GPGCMD} --yes --symmetric --passphrase "thisistosavethis" p.txt 1>>${strLogFile} 2>&1

rm -f p.txt

if [[ iCnt -eq 1 ]]; then

strCryptedFiles=${strFilesToProcess}.gpg

else

strCryptedFiles=${strTmp}

fi

}

#------------------------------------------------------------------------------

# Decrypt the Binary or ASCII GPG file.

#

# where input.txt contains:

# o passphrase (Eat my shorts)

# o an answer to the Question, "Output file 'filename' already exists,

# overwrite (y/n)?

# o a new plaintext filename

# or use FORCE to run PGP non-interactively.

# o When you decrypt a file that has a filename with the same name an

# another in the directory, FORCE causes PGP to overwrite the original

# file without prompting.

#------------------------------------------------------------------------------

procDecryptFiles()

{

if ${bDebug}; then

set -x

echo "Entering procDecryptFiles()" >> ${strLogFile}

fi

strDestFile=$1

typeset strTmp""

typeset -i iCnt=0

if [[ ! -f ${strPassDir}/p.txt.gpg ]]; then

procLogAbort 3 "Pass phrase file ${strPassDir}/p.txt.gpg not found"

fi

cd ${strPassDir} 1>>${strLogFile} 2>&1

${GPGCMD} --passphrase "thisistosavethis" --yes p.txt.gpg 1>>${strLogFile} 2>&1

cd ${strWorkingDir}

for j in ${strFilesToProcess}

do

chmod 660 ${j} 1>>${strLogFile} 2>&1

printf "Decrypting `ls -lF ${j}`" >> ${strLogFile}

strOutFile=`echo ${j} | cut -d. -f1` # Get the output filename

if [[ -z ${strDestFile} ]]

then

${GPGCMD} --output ${strOutFile} --decrypt --yes --passphrase-file \

${strPassDir}/p.txt ${j}

fi

if [[ ! -f ${strOutFile} ]]

then

cd ${strPassDir}

${GPGCMD} --symmetric --passphrase "thisistosavethis" --yes p.txt 1>>${strLogFile} 2>&1

rm -f p.txt 1>>${strLogFile} 2>&1

procLogAbort 3 "Unable to decrypt ${j}"

fi

iCnt=$(( ${iCnt} + 1 ))

chmod 660 ${strOutFile} 1>>${strLogFile} 2>&1

printf "Decrypted file at `pwd`:" >> ${strLogFile}

ls -lF ${strOutFile} >> ${strLogFile}

strTmp="${strOutFile} $strTmp"

done

cd ${strPassDir} 1>>${strLogFile} 2>&1

${GPGCMD} --symmetric --passphrase "thisistosavethis" --yes p.txt 1>>${strLogFile} 2>&1

rm -f p.txt 1>>${strLogFile} 2>&1

if [[ iCnt -eq 1 ]]; then

strCryptedFiles=${strOutFile}

else

strCryptedFiles=${strTmp}

fi

}

#==============================================================================

# When retrieving files from partners' websites we have to see if there are any

# new files posted to process.

#==============================================================================

procEchoNotFoundMessage()

{

if ${bDebug}; then

set -x

echo "Entering procEchoNotFoundMessage" >> ${strLogFile}

fi

echo "No files found to SSH ${strTrajectory} from ${strUser}@${strSSHAddress}." >> ${strLogFile}

if [[ -f ${strLSofDestDirFile}.0 ]]; then

echo "See previous log:" >> ${strLogFile}

ls -lF ${strLSofDestDirFile}.0 >> ${strLogFile}

echo "Which contains files '`cat ${strLSofDestDirFile}.0`'" >> ${strLogFile}

fi

echo "Finished processing of company ${strCompany} on `date`" >> ${strLogFile}

exit

}

#==============================================================================

# This routine is passed a string of files to 'get' or 'put' to a partner' SSH

# server. It is assumed the code calling this procedure has processed and

# vetted the files to be transferred. This procedure makes no decisions about

# which format these files are in or whether they should be transferred, it only

# attempts to do so to the data specified.

#

# The variables necessary to establish a connection to our partner's SSH server

# are checked to see if they have been set. Other variables such as the

# working directory where the files to be transferred are housed are also

# checked and assumed to be set up prior to calling this procedure.

#==============================================================================

procSSHFiles()

{

if ${bDebug}; then

set -x

echo "Entering procSSHFiles()" >> ${strLogFile}

fi

strDestFile=$1 # Sometimes we may want to specify the name of the destination file

if [[ -z ${strFilesToProcess} ]]; then

procEchoNotFoundMessage

else

# Make sure the necessary SSH variables are set:

if [[ -z ${strUser} ]]; then

procLogAbort 3 "No SSH user ID specified for ${strCompany}"

elif [[ -z ${strPass} ]]; then

procLogAbort 3 "No SSH password specified for ${strCompany}"

elif [[ -z ${strSSHAddress} ]]; then

procLogAbort 3 "No SSH address specified for ${strCompany}"

fi

fi

echo "Files being processed for ${strCompany} are ${strFilesToProcess}" >> ${strLogFile}

cd ${strWorkingDir}

if [[ ${strTrajectory} = "get" ]]; then

# Use the SSH scp command to copy the files to our account

#

typeset -i j

for i in ${strFilesToProcess}

do

(( j = j+1 ))

done

if [[ j -eq 1 ]]; then

if [[ -z ${strDestFile} ]]

then

echo "${strTrajectory}ing file ${strFilesToProcess}" >> ${strLogFile}

echo "From ${strUser}@${strSSHAddress}" >> ${strLogFile}

echo "scp ${strUser}@${strSSHAddress}:${strCompany}/out/${strFilesToProcess} ." >> ${strLogFile}

scp ${strUser}@${strSSHAddress}:${strCompany}/out/${strFilesToProcess} . 1>>${strLogFile} 2>&1

else

echo "${strTrajectory}ting file ${strDestFile}" >> ${strLogFile}

echo "From ${strUser}@${strSSHAddress}" >> ${strLogFile}

echo "scp ${strUser}@${strSSHAddress}:${strCompany}/out/${strDestFile} ${strDestFile}" >> ${strLogFile}

scp ${strUser}@${strSSHAddress}:${strCompany}/out/${strDestFile} ${strDestFile} 1>>${strLogFile} 2>&1

fi

else

echo "${strTrajectory}ting file ${strDestFile}" >> ${strLogFile}

echo "From ${strUser}@${strSSHAddress}" >> ${strLogFile}

echo "scp ${strUser}@${strSSHAddress}:${strCompany}/out/\* ." >>${strLogFile}

scp ${strUser}@${strSSHAddress}:${strCompany}/out/\* . 1>>${strLogFile} 2>&1

fi

elif [[ ${strTrajectory} = "put" ]]; then

echo "${strTrajectory}ting file(s) ${strFilesToProcess}" >> ${strLogFile}

echo "To ${strUser}@${strSSHAddress}" >> ${strLogFile}

echo "scp ${strFilesToProcess} ${strUser}@${strSSHAddress}:${strSubDir}" 1>>${strLogFile} 2>&1

scp ${strFilesToProcess} ${strUser}@${strSSHAddress}:${strSubDir} 1>>${strLogFile} 2>&1

else

procLogAbort 3 "Invalid trajectory specified as '${strTrajectory}' for ${strCompany}"

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

for i in ${strFilesToProcess}

do

chmod 660 ${i} 1>>${strLogFile} 2>&1

iRet=$?

if [[ iRet -ne 0 ]]

then # chmod failed

if [[ ! -f ${i} ]]

then

procLogAbort 3 "${strWorkingDir}/${i} does not exist after 'ssh ${strTrajectory}'"

else

id >> ${strLogFile}

ls -al . >> ${strLogFile}

procLogAbort 3 "chmod 660 on ${strWorkingDir}/${i} failed."

fi

fi

done

}

#==============================================================================

# Get a directory listing of our partner's website. We do this to see what,

# if any new files may have been posted on our clients SSH server. We store

# this listing in a file to compare to a previous directory listing to see if

# there are any files we need to retrieve and process.

#==============================================================================

procGetSSHLSofDest()

{

if ${bDebug}; then

set -x

echo "Entering procGetSSHLSofDest()" >> ${strLogFile}

fi

procStash ${strLSofDestDirFile}

# strCmd=`echo "ssh ${strUser}@${strSSHAddress} ""ls ${strSubDir}"""`

# strCmd=`echo "ssh ${strUser}@${strSSHAddress} ""find ${strSubDir} -maxdepth 1 -type f"""`

strCmd=`echo "ssh ${strUser}@${strSSHAddress} ""ls -l ${strSubDir} | awk '{print \$9}'"""`

echo "Executing '${strCmd}'" >> ${strLogFile}

strFilesToProcess=`${strCmd}`

echo ${strFilesToProcess} > ${strLSofDestDirFile}

if [[ -z ${strFilesToProcess} ]]; then

procEchoNotFoundMessage

fi

}

#------------------------------------------------------------------------------

# Call the appropriate procedure to decrypt or encrypt the file(s).

#------------------------------------------------------------------------------

procProcessCryptFormat()

{

if ${bDebug}; then

set -x

echo "Entering procProcessCryptFormat()" >> ${strLogFile}

fi

strCrypt=$1

case ${strCrypt} in

encrypt)

case ${strGPGFormat} in

Binary)

procEncryptFiles

;;

BinarySigned)

procEncryptAndSign

;;

ASCII)

procEncryptFiles

;;

ASCIISigned)

procEncryptAndSign

;;

\*)

procLogAbort 2 "Invalid GPG Format '${strGPGFormat}' specified..."

;;

esac

;;

decrypt)

case ${strGPGFormat} in

Binary)

procDecryptFiles

;;

BinarySigned)

procDecryptFiles

;;

ASCII)

printf "${strGPGFormat} not supported for decryption." | tee -a ${strLogFile}

exit 0

;;

ASCIISigned)

printf "${strGPGFormat} not supported for decryption." | tee -a ${strLogFile}

exit 0

;;

\*)

procLogAbort 2 "Invalid GPG Format '${strGPGFormat}' specified..."

;;

esac

;;

\*)

procLogAbort 2 "Invalid Crypt directive '${strCrypt}' specified..."

;;

esac

}

#==============================================================================

# Test Company is a make believe company that is used to test changes to this

# script. It is an account setup on our HCU/SBO SSH server that we can connect

# to and test SSH and decryption automation. This procedure will either download

# and decrypt files from our test company, or just decrypt files housed on our

# local SSH client in the 'strWorkingDir' directory.

#==============================================================================

procProcessTestCompanyGet()

{

if ${bDebug}; then

set -x

echo "Entering procProcessTestCompanyGet()" >> ${strLogFile}

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

if ${bPutGet}; then

procGetSSHLSofDest

if [[ -f ${strLSofDestDirFile}.0 ]]; then

strFilesToProcess=`diff ${strLSofDestDirFile} ${strLSofDestDirFile}.0`

fi

procSSHFiles

fi

procArchiveFiles

if ${bEnDeCrypt}; then # Decrypt files from partner website.

strFilesToProcess=`find . -maxdepth 1 -type f | grep -e gpg -e asc`

if [[ -z ${strFilesToProcess} ]]; then

echo "No files found to decrypt at ${strWorkingDir}." >> ${strLogFile}

fi

echo "Company ${strCompany} files ${strFilesToProcess} will be decrypted." >> ${strLogFile}

procProcessCryptFormat decrypt

fi

}

#==============================================================================

# See if there are files in the working directory that need to be encrypted.

# If so, encrypt and move the original files to the archive directory for

# reference later if that becomes necessary.

#==============================================================================

procProcessTestCompanyPut()

{

if ${bDebug}; then

set -x

echo "Entering procProcessTestCompanyPut()" >> ${strLogFile}

fi

cd ${strWorkingDir} 1>>${strLogFile} 2>&1

if ${bEnDeCrypt}; then

if ! ${bFileSpecified}

then

strFilesToProcess=`find . -maxdepth 1 -type f | grep -v gpg | grep -v asc`

fi

if [[ -z ${strFilesToProcess} ]]; then

procEchoNotFoundMessage

fi

printf "Company ${strCompany} files ${strFilesToProcess} being decrypted." >> ${strLogFile}

procCheckFiles

procProcessCryptFormat encrypt

echo "Company ${strCompany} files encrypted." >> ${strLogFile}

# Archive the orignal files that have been encrypted

procArchiveFiles

fi

if ${bPutGet}; then

# Files in 'out' directory need to be transferred to partner SSH server.

if ! ${bFileSpecified}; then

strFilesToProcess=`find . -maxdepth 1 -type f`

fi

procSSHFiles

fi

procRemoveFiles

}

#------------------------------------------------------------------------------

# Set all company specific variables based on command line arguments.

#------------------------------------------------------------------------------

procSetCompanyVars()

{

if ${bDebug}; then

set -x

printf "Entering procSetCompanyVars()" >> ${strLogFile}

fi

if [[ -z ${strLogsDir} ]]; then

strLogsDir=$HOME/logs

fi

if [ ! -d ${strLogsDir} ]; then

procLogAbort 1 "Dir-> ${strLogsDir}\nDoes not exist, attempting to create."

mkdir ${strLogsDir}

if [[ $? -ne 0 ]]

then

procLogAbort 3 "Could not create:\nDir-> ${strLogsDir}."

fi

chmod 770 ${strLogsDir}

fi

case ${strCompany} in

lifeinsurance)

if ${bTestMode}

then

strEmailList="root@localhost"

else

strEmailList="admin1@domain.com,admin2@domain.com"

fi

strTrajectory=put

strWorkingDir=${strPayOutDir}

strCoGPGKeyID="0x8EEB4512"

strUser=""

strPass=""

strGPGFormat="Binary"

strSSHAddress=""

strSubDir="."

strLogFile="${strLogsDir}/life\_${strBaseName}.log"

strLSofDestDirFile=""

;;

legalservices)

if ${bTestMode}

then

strEmailList="root@localhost"

else

# On failure conditions send log to:

strEmailList="admin1@domain.com,admin2@domain.com"

fi

strCoGPGKeyID="0x9A3291D5"

strUser=sshlegal

strPass="legalpass" # New password as of 10 Nov. 2005

strSSHAddress="65.199.74.23"

strGPGFormat="Binary"

strSubDir="."

case ${strTrajectory} in

get)

strSubDir="company"

strWorkingDir=$HOME/legal/in

strLogFile="${strLogsDir}/legal\_get\_${strBaseName}.log"

strLSofDestDirFile="${strLogsDir}/legal\_get\_${strBaseName}\_ls.log"

;;

put)

strWorkingDir=$HOME/legal/out

if ${bTestMode}

then

strSubDir="legaltest"

else

strSubDir="mybusiness"

fi

strLogFile="${strLogsDir}/legal\_put\_${strBaseName}.log"

strLSofDestDirFile=""

;;

\*)

procLogAbort 2 "Invalid Trajectory '$strTrajectory' specified..."

;;

esac

;;

benefitservices)

if ${bTestMode}

then

strEmailList="root@localhost"

strSubDir="bentestdir"

else

strEmailList="admin1@domain.com,admin2@domain.com"

strSubDir="bendir"

fi

strTrajectory=put

strCoGPGKeyID="0x5EBB4182"

strUser=sshyourcompany

strPass="benefitpass"

strSSHAddress="42.266.29.34"

strGPGFormat="Binary"

strWorkingDir=$HOME/benefits

strLogFile="${strLogsDir}/benefits\_${strBaseName}.log"

strLSofDestDirFile="${strLogsDir}/benefits\_${strBaseName}\_ls.log"

strSSHCMDFile="${strLogsDir}/benefits\_${strBaseName}.ssh"

;;

payrollservices)

if ${bTestMode}

then

strEmailList="root@localhost"

strSubDir="paytestdir"

else

# Failure email list only

strEmailList="admin1@domain.com,admin2@domain.com"

strSubDir="paydir"

fi

strSSHAddress="SSH.BENEFITS.COM" # 62.333.45.67

strCoGPGKeyID="0x2B235675"

strUser="payuser"

strPass="paypass"

strGPGFormat=Binary # use -e to encrypt

case ${strTrajectory} in

get)

strSubDir="outgoing"

strWorkingDir=${strBenInDir}

strSSHCMDFile="${strLogsDir}/pay\_get\_${strBaseName}.ssh"

strLogFile="${strLogsDir}/pay\_get\_${strBaseName}.log"

strLSofDestDirFile="${strLogsDir}/pay\_get\_${strBaseName}\_ls.log"

;;

put)

if ${bTestMode}

then

strSubDir="incoming/mytest"

else

strSubDir="incoming"

fi

strWorkingDir=${strBenOutDir}

strSSHCMDFile="${strLogsDir}/pay\_put\_${strBaseName}.ssh"

strLogFile="${strLogsDir}/pay\_put\_${strBaseName}.log"

strLSofDestDirFile=""

;;

\*)

procLogAbort 2 "Invalid Trajectory '$strTrajectory' specified..."

;;

esac

;;

localsshserver)

if ${bTestMode}

then

strEmailList="root@localhost"

else

strEmailList="admin1@domain.com,admin2@domain.com"

fi

strCoGPGKeyID="0xE403569A"

strSSHAddress="xxx"

strUser=sshuser

strPass=sshpass

strWorkingDir=$HOME/ssh/in

if [[ -z ${strGPGFormat} ]]; then

strGPGFormat=Binary

fi # Can be specified on command line

strSubDir="."

strLogFile="${strLogsDir}/ssh\_get\_${strBaseName}.log"

strLSofDestDirFile="${strLogsDir}/ssh\_get\_${strBaseName}\_ls.log"

strSSHCMDFile="${strLogsDir}/ssh\_get\_${strBaseName}.ssh"

strSSHLSFile="${strLogsDir}/ssh\_get\_${strBaseName}\_ls.ssh"

;;

testcompany)

strCoGPGKeyID="0x11C12F2E" # Test Company public key

strSSHAddress=thecaptainslatest.dyndns.org

if [[ -z ${strGPGFormat} ]]; then

strGPGFormat=Binary

fi # Can be specified on command line

strUser=testuser

strPass=nopass # Key validation only

case ${strTrajectory} in

get)

strWorkingDir=${HOME}/ssh/in

strSubDir="testcompany/out"

strLogFile="${strLogsDir}/ssh\_get\_${strBaseName}.log"

strLSofDestDirFile="${strLogsDir}/ssh\_get\_${strBaseName}\_ls.log"

;;

put)

strWorkingDir=${HOME}/ssh/out

strSubDir="testcompany/in" # Partner subdirectory

strLogFile="${strLogsDir}/ssh\_put\_${strBaseName}.log"

strLSofDestDirFile=""

;;

\*)

procLogAbort 2 "Invalid Trajectory '$strTrajectory' specified..."

;;

esac

;;

\*)

Usage

procLogAbort 2 "Invalid company name '${strCompany}' specified..."

;;

esac

if [ ! -d ${strLogsDir} ]; then

procLogAbort 1 "Dir-> ${strLogsDir}\nDoes not exist, attempting to create."

mkdir -p ${strLogsDir}

if [[ $? -ne 0 ]]; then

procLogAbort 3 "Could not create:\nDir-> ${strLogsDir}."

fi

chmod 770 ${strLogsDir}

fi

if [[ ! -d ${strWorkingDir} ]]; then

procLogAbort 1 "Dir-> ${strWorkingDir}\nDoes not exist, attempting to create."

mkdir -p ${strWorkingDir} 1>>${strLogFile} 2>&1

if [[ $? -ne 0 ]]; then

procLogAbort 3 "Could not create:\nDir-> ${strWorkingDir}."

fi

chmod 770 ${strWorkingDir} 1>>${strLogFile} 2>&1

fi

if [ ! -d ${strWorkingDir}/archive ]; then

procLogAbort 1 "Dir-> ${strWorkingDir}/archive\nDoes not exist, attempting to create."

mkdir ${strWorkingDir}/archive 1>>${strLogFile} 2>&1

chmod 770 ${strWorkingDir}/archive

fi

}

#==============================================================================

# MAIN PROGRAM:

# - Perform initial error checking

# - Setup company specific variables

# - Initialize log file

# - Determine process to perform on company being processed

#==============================================================================

#

# Check to see if number of parameters is correct... may be needed in

# future.

#

if [ "$#" -eq 0 ]

then

Usage

procLogAbort 2 "No Argument specified..."

fi

procSetCompanyVars

# It is assumed that files you are receiving from customers is known and

# should not be altered.

if [[ ${strTrajectory} = "get" ]]

then

if ${bFileSpecified}

then

procLogAbort 2 "All file gets cannot have the files to be processed overridden."

fi

fi

#procStash ${strLogFile} # Stash the previous log and initialize a new log

if [[ -f ${strLogFile} ]]; then

rm ${strLogFile}

fi

echo "Started processing of company ${strCompany} on `date`" > ${strLogFile}

procLogVars

case ${strCompany} in

lifeinsurance)

case ${strTrajectory} in

get)

procProcessLifeGet

;;

put)

procProcessLifePut

;;

\*)

procLogAbort 2 "Invalid Trajectory '${strTrajectory}' specified..."

;;

esac

;;

legalservices)

case ${strTrajectory} in

get)

procProcessBNYGet

;;

put)

procProcessBNYPut

;;

\*)

procLogAbort 2 "Invalid Trajectory '${strTrajectory}' specified..."

;;

esac

;;

benefitservices)

procProcessBenefitPut

;;

payrollservices)

procPayrollPut

;;

localsshserver)

case ${strTrajectory} in

get)

procProcessLocalGet

;;

put)

procProcessLocalPut

;;

\*)

procLogAbort 2 "Invalid Trajectory '${strTrajectory}' specified..."

;;

esac

;;

testcompany)

case ${strTrajectory} in

get)

procProcessTestCompanyGet

;;

put)

procProcessTestCompanyPut

;;

\*)

procLogAbort 2 "Invalid Trajectory '${strTrajectory}' specified..."

;;

esac

;;

\*)

Usage

procLogAbort 2 "Invalid company '${strCompany}' specified..."

esac

echo "Finished processing of Company ${strCompany} on `date`" >> ${strLogFile}

## Encrypting Emails Before Sending Them over the Infected Internet -- Don’t Expose Your Correspondence to Crackers Everywhere

If you do not understand how email works, your transmission of an email bounces from Mail Transfer Agent (MTA) to MTA server until it reaches its final destination. An MTA is simply a computer server maintained by someone that relays your unencrypted email, which may be intercepted and read, manipulated, and/or exploited. Any documents attached may also receive the same treatment. This is similar to a "[Man-in-the middle](file:///F:\Kirk\Book\Man-in-the%20middle)" attack. From, <https://en.wikipedia.org/wiki/Email>, "*Email privacy, without some security precautions, can be compromised because:*

* *Email messages are generally not encrypted.*
* *Email messages have to go through intermediate computers before reaching their destination, meaning it is relatively easy for others to intercept and read messages.*
* *Many Internet Service Providers (ISP) store copies of email messages on their mail servers before they are delivered. The backups of these can remain for up to several months on their server, despite deletion from the mailbox.*
* *The "Received:" fields and other information in the email can often identify the sender, preventing anonymous communication.*"… and from other sources:
* Hackers have stolen the login credentials from many Internet users and can access their email anytime.
* Leaked classified documents show that that NSA has direct access to encrypted emails stored on various U.S. Corporations servers, which they deny. The NSA also makes copies of those emails on their own servers for years of reference.
* The NSA has obtained direct access to the systems of Microsoft, Google, Facebook, Apple, Yahoo, AOL, YouTube, Skype and PalTalk, as well as other U.S. giants, which they also deny.

(See: <http://en.wikipedia.org/wiki/Man-in-the-middle_attack>, <http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data>, <http://news.softpedia.com/news/Microsoft-Offered-the-NSA-Direct-Access-to-Outlook-com-Skype-SkyDrive-Accounts-Report-367531.shtml>)

You must encrypt or password-protect any email sent over the Internet, **PERIOD**. Think about it, would you put a letter in your post office mailbox without an envelope? Crackers scan and record unencrypted email, text messages, and attachments for things such as passwords and usernames. Crackers will steal this information and use it for their own gain. Corporations and governments will record this information for years, which may come back to haunt you. If you want to send a file, please get help from a friend or neighbor on how to, at the very least, password-protect or encrypt that file. These precautions are the bare minimum of protection that you should be using for sending data on our infected Internet.

Every email you send unencrypted goes through multiple MTAs and are read going into and out of those servers. Crackers, your ISP, everyone will make a copy of your email message and your attachments on their server's hard drives. Some will go through everything that passed through their servers looking for things of value like usernames, passwords and things in attachments. They also collect email addresses and sell them to spammers everywhere. People everywhere send email unencrypted with valuable information that crackers, governments and others exploit. Assume that every email you sent unencrypted HAS BEEN READ by someone else and examined for exploitation.

So the answer is NEVER, EVER send anything of value in an unencrypted email! The traditional method of sending encrypted email requires that the person you’re sending the email to, have everything set up to receive an encrypted email. This requires an exchange of public keys with the person to whom you are sending that encrypted email. You both will have to develop the skills necessary to generate, exchange keys, encrypt and decrypt your correspondence. This is not very complex, and with a few simple directions, you can make this happen, which we will soon cover.

* You want to send a friend an encrypted email. To do so you have to generate a public/private key pair and send your public key to your friend. You then have to instruct your friend on how to generate a public/private key pair and have them send you their public key.
* Once these keys have been exchanged, and in order to exchange encrypted messages, you both have to encrypt your data to your public keys, preferably using both at the same time so you can both decrypt the received and sent messages.

The above process is not complex, and you may have some business partners who resist learning how to do this. For example, I have tried to get friends and family to encrypt email and found little interest from them in learning how to do so. I have to be very careful with any correspondence with them as a result. However, if you are an SBO, you should not do business with a partner that will not learn how to encrypt their correspondence.

### Encrypting Email with Outlook and Encipher.it

Outlook encryption is key-based encryption. The email account user has to generate a digital certificate and exchange that certificate via a digitally signed email message with the Outlook user whom they wish to correspond with and vice versa. The certificates are then used to automatically encrypt and decrypt email correspondence between the two recipients. To set this up, open Outlook and read <http://office.microsoft.com/en-us/outlook-help/get-a-digital-id-HP010355070.aspx>. Be careful using this form of email encryption, as there is abundant evidence that the NSA has backdoors in most U.S. corporate encrypted email.

***Then in Outlook when sending an encrypted email click on Options > More Options > Security Settings > check the box Encrypt message contents and attachments***.

Encipher.it boldly states they will protect your email from being read by hackers, eavesdroppers, NSA, your employer or your with. Their privacy policy is outstanding and says "*Any personal information received will only be used to fill your order. We will not sell or redistribute your information to anyone.*" Encipher.it uses JavaScript in your browser to encrypt your message so no data leaves your browser in plain text protecting it from packet sniffing software.

Encipher.it is a website that will allow you to encrypt text to a password and send that encrypted message in an email. From their website to use their encryption:

1. Add a bookmark in Firefox or Google Chrome:

* Drag this link: [https://encipher.it](https://encipher.it/) to your Bookmarks Bar

Internet Explorer:

* Right click: Encipher It and click on "Add to Favorites"

1. Encrypt the message

* Login to Google Mail, Facebook, or another site where you want to protect the text.
* Type your message and click the bookmark.
* Enter the password and click the "Encrypt" button.

1. Decrypt the message

* To decrypt the message on the page, just click bookmark again and reenter the same password.

What <https://encipher.it> does is encrypt your message to a password that you can send, phone, or text to the person to whom you send your email message to. The recipient of the message will have to visit the Encipher.it website or add the bookmark to be able to decrypt your message. According to an article written by Alex Wawro from PCWorld titled "<https://encipher.it> Encrypts Email for Free", July 2012, "*The bookmarklet runs all code locally on your PC, so there's no danger of Encipher.it staff or someone else listening in as you transmit messages to a server for encryption/decryption. The Encipher add-on will apply a 256-bit Advanced Encryption Standard (AES) algorithm,* [*https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard*](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) *that runs in real time; 256-bit encryption is the same kind of data protection that the websites of many banks and government services employ.*"

To encrypt your email using a browser simply type the message in your text editor or word processor and paste it on the website to encrypt it. ***Surf to*** [*https://encipher.it*](https://encipher.it/) ***and copy your message to the clipboard > in the Type the message here to encrypt it box paste or type in your message > click on the Encipher It button and next to Encryption Password enter your encryption password > you can then click on Send to Gmail or Copy button to send it to another email service provider > when your recipient gets your message if they are using a web browser to view their email they can use the Encipher-it plug-in to decrypt it using the password you provide them > if they are using a local email server visit the website and cut and paste the message > click on the*** [*Encipher It*](https://encipher.it/) ***link > then enter the password that you provided them to decrypt the message***.

***To encrypt you email in Linux surf to and click on Download > choose the Linux button and download the encipher\_2.3\_amd64.deb file to your Downloads directory > become root and type the following on the command line in Debian Linux flavors:***

# cd ~suroot/Downloads

# dpkg -i encipher\_2.3\_i386.deb

# dpkg -i encipher\_2.3\_amd64.deb

# apt-get update && apt-get upgrade -y

To make all this work for any browser visit the <https://encipher.it/email-encryption#install> web page and ***under Bookmarklets on the left next to Drag this link:, drag the Encipher It link to the Bookmarks Bar in Firefox or Google Chome***. For Internet Explorer right click on the link and click Add to Favorites. Type in your message to your browser email and click on the Encipher It bookmark and a dialog box will appear asking you to Enter encryption key which is the passphrase you will need to provide to your message recipient.

### Use Multiple Email Addresses for Your Correspondence

Don't allow advertising to brainwash you into believing that Google, Microsoft, ISPs and other big corporations that record every email you send and share with them and the NSA are your only choices for email. The truth is if you look around you can find FREE email providers that actually want to protect your privacy. We should use these services to safeguard all of the various facets of our lives online. We can use them to set up separate email addresses for all of the things we do in life. For years I have maintained a few different email addresses that I have used for various things. I have a Microsoft Hotmail address, which I have used since the Internet was born, which I log into, that allows all of the big spamming corporations and crackers to sell this email address and spam it constantly (I hope that spam costs Microsoft, crackers and the NSA many thousands in storage on their servers). Of course, as a U.S. citizen, these are my taxpayer dollars. My ancient Hotmail email address gets upward of 50 to 200 spam emails a day. Since we all now know that these U.S. corporations share everything we do on the Internet, I think we should record our Microsoft and Google Email address everywhere and allow everyone to spam and record everything we do, thus littering their servers with things we care nothing about. This is a definitive way that we can all fight back against this invasion into our privacy.

Something else for you to consider is when you use the email services from the major ISP providers, they keep all of your email for years, and in the USA, the NSA has been storing your Internet activity since 2001. U.S. citizens are just now coming to know these facts and are complacently going along with this mass monitoring of their Internet activity by corporations and their government, which I find astonishing. Additionally, their ISPs scan their emails to create profiles of their individual lives. This information is then freely available to anyone who wants to buy it. (E.g. An inflammatory email written 10 years ago can be used to investigate anyone communicating with a U.S. citizen, deny you or them of insurance, or even a job). This level of scrutiny does not seem to register with the average U.S. citizen as to how adversely it could affect their everyday lives. Nevertheless, there is little outcry from U.S. citizens about this invasion into their privacy, something generations of soldiers and myself fought to prevent.

Let's develop separate sets of email accounts, one for shopping logins, another for banking and investing, and at least a third for other things such as your social media accounts. A fourth could be used to develop a set of very private accounts only shared between a few trusted sources:

1. You want a very public email address that all the spammers, your ISP and government know about. Register this email address at every shopping or government site you use, and give this email address to friends and relatives that also send spam. For example, register with CDW, Domino's Pizza, Dell, Federal Student Aid, Linux Professional Institute, Pizzahut, Best Buy, NewEgg, GM, GroupGolfer, GolfNow, Honda, LastMinuteGolf, Logitech, RedHat, REI, Soaring Eagle Casino, TigerDirect, Travelocity, USGS Store, VMware, Walmart, etc.. Visit it every so often to flush it out and make it look like this is your very active primary email account. Never tie this email address to an HC/SB domain/website or blog for logins, never use or register this email account at any financial institution you do business with.
2. Create a job search or SBO email address and try to keep it as private as possible. Unfortunately, as you put it up on LinkedIn and post on job search websites, it will become a spam haven also. You also have to provide this email address on applications, resumes, business cards and employer websites who will spread your job search email address to third parties. Nevertheless, to exist in today's information-sharing world you have to allow this email address to be compromised. It goes without saying, use this email address as professionally as possible, never conduct personal or clandestine matters, never communicate with friends and family using this email address, and never register or use it with a financial institution.
3. There are people we trust in life for whom we should maintain a separate email address for unencrypted and/or encrypted email. This would be our very private email address that we only hand out to a few select individuals. The "Hide My Ass" and "Anonymouse" services, discussed in [Chapter 8](#_Use_a_Web), offer free anonymous email accounts. There are also other notable, somewhat free email services with good privacy protections, which include Hushmail, Silentcircle and Unspyable. For example, Hushmail will give you a free email account, but you cannot configure an email client such as Thunderbird, and you must log in every three weeks to keep the account. POP & IMAP are available only to Hushmail Premium customers with Desktop Access and Hushmail Business with Desktop Access subscribers. (See: [https://countermail.com](https://countermail.com/), [https://www.hushmail.com](https://www.hushmail.com/), <https://riseup.net>, [https://silentcircle.com](https://silentcircle.com/), <http://unspyable.com>, <https://help.hushmail.com/entries/243624-thunderbird-3-imap>)

|  |  |
| --- | --- |
| **WARNING** | Hushmail promises "end-to-end" encryption, which means their keys can be used to decrypt your email. Refer forward to how Lavabit was shutdown as the U.S. government requested their private keys. True end-to-end encryption means that the service provider cannot look at your communications even if they wanted to. Riseup requests that you have cookies enabled, which is suspicious, so you have to question their email service as well. |

1. Maintain a separate email account for financial matters that you check on often. This is the email account that we use to login at our credit unions, banks, legal firms, PayPal, investment firms and shopping sites that you allow to maintain financial data such as a registered credit card. There are two trains of thought on this account. You may want it as public as possible using Gmail, Outlook or Yahoo so that if your identity is stolen there are records of correspondence with your financial institutions that are kept by your ISP, email provider and the NSA, or perhaps you want a private email service such as Hushmail to keep your financial correspondence as private as possible between you and your financial institutions. Realize that any email correspondence with your financial institutions is only private on your side of the fence and that your financial institution may keep your correspondence for their records.
2. You may want an encrypted email address that you can use to correspond with your business partners or people of like mind that feel your correspondence need not be read by everyone on the Internet. This might be one of the email addresses above, but I have found it easier to have a separate email address for encrypted email. You can also use this email account for your website/domain and blog, and perhaps social media sites if they are used for business purposes. For example, some Twitter handles have monetary value and have huge numbers of followers. Never use this email as a login to a financial institution or for unencrypted email.
3. You can also create a disposable email address at [http://maildrop.cc](http://maildrop.cc/). MailDrop offers up a quick disposable email address so that you can register at websites you want to access that want your personal information. There is no privacy protection using MailDrop, as anyone can read anything you send or receive, but sometimes that is not a concern. MailBox is by design a layer of obfuscation to using your real email addresses that get spammed constantly. While your email is stored for posterity and government viewing in a MailDrop, your Internet activity is not tracked by MailDrop. MailDrop is just another tool in your privacy arsenal to consider.

|  |  |
| --- | --- |
|  | Historical Reference – Once upon a time there was Lavabit to protect the privacy of U.S. Internet users |
| Please visit [http://lavabit.com](http://lavabit.com/), which was a private email service run by Ladar Levison, Owner and Operator of Lavabit, LLC. It now hosts his testimony as to how invasive the U.S. government surveillance system has become. His company was forced out of business by the U.S. government as the FBI relentlessly pursued Edward Snowden. Edward hosted his email account with Lavabit, and Lavabit was subsequently served with a U.S. national security letter in which the FBI demands information about a user, but the email service provider was prohibited to tell his users or anyone else about this request. Ladar tried to resist this and sent five, 2,560-character SSL encryption keys on an 11-page printout in what was more or less, an illegible 4-point type. To decrypt Snowden's emails, someone at the FBI would have had to type every character perfectly. However, a U.S. court ordered a $5,000-a-day fine on Lavabit LLC, and on Aug 7, 2013, Ladar handed over the digital copies of the keys. Ladar shutdown his business with a letter stating "*This experience has taught me one very important lesson: without congressional action or a strong judicial precedent, I would \_strongly\_ recommend against anyone trusting their private data to a company with physical ties to the United States.*" Among other things, I encourage you to read his full statement. Ladar was later found in contempt of court for resisting turning over his Secure Socket Layer key that was used to encrypt communications for his 400,000 users. This key would have allowed the U.S. government to access the communications of all of Lavabit's users, something that the U.S. Fourth Amendment used to protect.  After the U.S. government released Ladar Levison from a criminal sentence for silence order, he explained to his very angry email service users exactly why he had suddenly shutdown access to all of his Internet email users without allowing them to backup their email. If he had warned them that he was shutting down his email service, the U.S. Government had the right by their new invasive laws, to demand that he keep the site up and running while the U.S. government, who now had the encryption keys, could/would monitor/compromise all of their communications for all time. He could not, in good conscience, allow that to happen.  He has endured a lot of criticism and anger about this decision, but he said he would bring the site back up if the U.S. government ever allowed him to generate a new set of uncompromised keys that he does not have to share. I admire his stand, and I hope others will follow his example. However, that is not the trend that other U.S. companies are following. The data leaked by Edward Snowden indicates that the NSA is maintaining a database of encryption keys used by many online providers who say they secure your data. These keys are obtained via court orders, buying the keys with taxpayer dollars, or by hacking into corporate networks and servers. Once the NSA has these keys they can easily monitor all Internet traffic flowing to and from these corporations.  Ladar later appealed and lost this ruling so just try to fathom how far the U.S. government is willing to go to invade their citizens' privacy with this type of behavior. Who wants to do business with a country willing to take such extreme measures? On May 4, 2014, Ladar released a final statement which sums up how secretive and heavy handed the U.S. government has become to invade their citizens' privacy. I think that any business respecting privacy and Internet freedom will have to move to move their facilities to legal jurisdictions, such as Germany, that provide much better protections for privacy and security. If you visit <http://lavabit.com> you can read Mr. Larders final salvo in his cry for the American people to wake up and fight for their freedom. Below are a few paragraphs of what he had to say.  "*When the judge granted the contempt charge unopposed – ignoring my attorney’s request to dispute the government’s claims – he created a loophole. I was never given an opportunity to object, let alone provide a meaningful defense. An important point, since the contempt charge endorsed new legal claims – reversing what the court had previously indicated. Without an objection on the record, the appellate court would rule that my right to an appeal had been waived – since the charges hadn’t been disputed in district court. Given the Supreme Court’s tradition of declining to review cases decided on procedural grounds, I will likely be denied justice, forever.*  *The most important question raised by my appeal was what constitutes a "search," i.e., whether law enforcement may demand the encryption keys of a business and use those keys to inspect the private communications of every customer, when they are only authorized to access information belonging to a select few.*  *The problem here is technological: until a communication has been decrypted and the contents parsed, it is impossible for a surveillance device to determine which network connections belong to the targeted accounts. The government argued that since the "inspection" would be carried out by a machine, they were exempt from the normal search-and-seizure protections of the fourth amendment.*  *More importantly, the prosecution argued the exemption was because my users had no expectation of privacy, even though the encryption they were trying to break was created specifically to ensure a users' privacy.*  *If my experience serves any purpose, it is to illustrate what most already know: our courts must not be allowed to consider matters of great importance in secret, lest we find ourselves summarily deprived of meaningful due process. If we allow our government to continue operating in secret, it is only a matter of time before you or a loved one find yourself in a position like I was – standing in a secret courtroom, alone, and without any of the unalienable rights that are supposed to protect us from an abuse of the state’s authority.*  *Sincerely,*  *Ladar Levison*  *Owner and Operator, Lavabit LLC*" | |

My advice is, if you want to try to keep your email somewhat private on the Internet, make sure you are dealing with a non-U.S. email service provider that I present in this book later, or encrypt your email yourself, which we will soon cover. You can use Microsoft, Google, Yahoo and others as your public email addresses, which we all now know freely provide information to the U.S. and British governments, or you can try Countermail, HushMail, Silentcircle, RiseUp and Unspyable for your more private correspondence. However, remember that email sent unencrypted is subject to observation by everyone connected to the Internet. However, the listed services above have greater privacy protection than the large U.S. corporations and ISPs. However, if you are a reader of forums and in tune with the IT buzz, many people suggest that the only true method of secure communication is to set up a GPG key exchange and encrypt email to you and your partner's public and private keys.

### Windows 7 -- Using Gpg4win and Claws to Encrypt Emails & Files

As you now know, any email sent via any major email provider is screened automatically by intelligence services. Most people in the U.S. I have met say "so what, let them screen me because I have nothing to hide". This is because they and their ancestors never experienced the Stasi excesses during Communist rule. A common joke in Germany is "Why, despite all the shortages, is toilet paper in eastern Germany two-ply? Because they have to send a copy of everything they do to Russia." German courts are blocking the implementation of the EU Data Retention Directive, and they have ruled against Google's data gathering practices. As German Chancellor Angela Merkel stated "*These two values – freedom and security – to a certain extent are and always have been at odds with one another. The proper balance needs to be struck again and again by means of the law. The end does not justify the means. Not everything that is technically possible should also be permissible.*" Western citizens in the U.S. do not understand how violations in their privacy can be exploited or how providing this information can come back to haunt them.

Gpg4win is Windows encryption software supported by the German Federal Office for Information Security (BSI). Because of the expressed German outrage over the NSA scandal, we can have a high degree of certainty that there are no U.S. backdoors built into this open source software. However, the source code is freely available, and you can rest assured it has been studied and perhaps broken by some very smart NSA folks. Gpg4win is actually a suite of encryption utilities bundled together that will be installed on your Windows computer. From the Gpg4win website those utilities are:

* GnuPG - The core; this is the actual encryption tool.
* Kleopatra - A certificate manager for OpenPGP and X.509 (S/MIME) and common crypto dialogs. (See: <http://docs.kde.org/stable/en/kdepim/kleopatra/kleopatra.pdf>)
* GPA - An alternative certificate manager for OpenPGP and X.509 (S/MIME).
* GpGOL - A plugin for Microsoft Outlook 2003/2007/2010/2013 (email encryption).
* GpgEX - A plugin for Microsoft Explorer (file encryption).
* Claws Mail - A complete email application with crypto support.
* Gpg4win Compendium - The documentation (for beginner and advanced users), available in English and German.

The Gpg4win Compendium documentation is an excellent reference for learning about their software, how email message encryption works and much more. Their encryption software is designed to integrate well with Microsoft Outlook using their GpGOL plugin. However, using other email clients is a bit more difficult, but easily mastered. Since many of you are probably not using Outlook, I decided to only present a simple solution using the Gpg4win Claws Mail client. Rather than repeat what is in Gpg4win's documentation, the following example is intended as a quick step-by-step guide to get you started. I chose to present only OpenPGP message encryption in Gpg4win because by using this method you can practice generating key pairs exchanging your public key with Gpg4win's automated practice server. I have found that many people are intimidated by this technology, and having a tool where you can practice without repercussions should make things fun and easy. For example, posting a public key on a public key server or saying something incorrect on technical forum can invite some hostile criticism and attacks from knowledgeable computer professionals. I highly encourage you to read Gpg4win manual at <http://www.gpg4win.org/doc/en/gpg4win-compendium.html>, which details everything you need to know about encrypting messages, files, setting up certificates and much more. If you are a SBO you will need to study and pay close attention to the certificates portions of the manual. Setting up, registering and using certificates is a somewhat complex topic that I chose not to cover in this book. It is important to an SBO, but has little or no significance to an HCU. (See: <https://en.wikipedia.org/wiki/Gpg4win>)

As previously discussed about encrypting files, and as you will learn about in detail in [Chapter 9](#_-_Hardening_security), using the digital communication encryption technology SSH, encrypting messages is very similar. Once you master any one of these technologies you know how they all work. The only problem is getting your business partners, friends, relatives and anyone else that we communicate with to take a few minutes and learn about them as well. As you will see, setting up your computer to encrypt messages is easy and hopefully fun for you. It is certainly important to your privacy and worth a few minutes of your time.

Just like you have a key or digital code to gain entry to your house or business, we have to create a key pair to encrypt the email messages that we will send and receive to/from our partners. As we discussed previously, while encrypting files, one of the keys in that pair is a public key that we will provide to our partners with whom we want to exchange encrypted messages. We will backup the private key that we generate and will never share or allow it to be compromised. In other words, it should never be housed on a computer or virtual machine that you use to surf the Internet or read questionable email. To get started using Gpg4win ***click on Start > All Programs > Gpg4win > select Kleopatra which we will now use to generate our key pair > click on the File menu upper left > select New Certificate… > choose Create a personal OpenPGP key pair > under Enter Details enter a bogus Name: John Doe, for Email: a valid email address that you created earlier for spam, and next to Comment: Test , Next > review what you entered and click on the Create Key button > enter a simple Passphrase (we are just practicing), re-enter your passphrase, OK > type in your random keyboard information and the Key Pair Successfully Created screen will appear along with your public keys Fingerprint > cut and paste your Fingerprint text file and store it the same directory where you plan to backup your key pair > click on Make a Backup Of Your Key Pair… and save both a GPG binary copy and a ASC copy of your keys in the Fingerprint directory naming them after the email address you used, Finish > this is a good time to generate a public key certificate that we will exchange with the Adele email robot to practice correspondence. Select your certificate and click on the File menu > select Export certificates… and save your public key to the directory you want, save it in both the binary (.gpg) and ASCII (.asc) forms***. The directory I chose to use for all of these files was on my backup USB drive because I have multiple computers from which I use to send encrypted email. Your OneDrive is also a good candidate for your public key so you can share it among your Internet devices. Since you will exchange these files with associates you want to share encrypted email with, consider naming you public key file after your public key's Fingerprint. This will emphasize to your partners how much they need to call and verify the fingerprint of your public key, which you will find they often ignore.

We have now generated and backed up our key pair. Now we have to configure Claws Mail to send and receive email. You can skip this step if you have Outlook or want to use another email client. For example, we will soon cover how to do this in Thunderbird, which is an alternative and perhaps superior email client to using Claws. To set up any email client you will need information from your email service provider, which you should have recorded when you originally setup your email account. I keep all this information in my Keepass Password Safe database, which we will talk about later. ***Click on Start > All Programs > Gpg4win > Claws Mail and the Welcome to Claws Mail wizard will appear, Forward > enter Your name:, Your email address:, and Your organization: (e.g. That Cyber Security Guy), Forward > you have to be careful on the Receiving mail dialog as the default Server type: POP3 is presented***. Most email services will support POP3, which will download and delete all email from your email service provider. Try changing this setting to IMAP, as most email services such as Gmail or Outlook support leaving email on their servers. If your setup using IMAP does not work, change the setting to POP3, which works with older ISPs that do not support IMAP. ***Enter your Server address:, Username: and Password:, Forward > on the Sending mail dialog enter your outgoing SMTP server address:, Forward > on Configuration finished, Save***. We are now ready to exchange public keys with the Gpg4win server robot and learn how to encrypt and decrypt email.

The Gpg4win project provides Adele, who is a very nice email robot with which you can set up and practice encrypted email correspondence. We must do this so that we don't embarrass ourselves trying to exchange keys with our partners or putting pubic keys on key servers before we know what we are doing. The Adele robot provides us the means to practice setting up and using email encryption technology. To set up an encrypted exchange of email or files, we have to exchange keys with our partners, or in this case, the Adele robot. Startup whichever email program you are using and in the subject type: "My Public OpenPGP Certificate" and either cut and paste your ASCII key in the email text body, or use the easier solution and just attach the <file>.gpg binary version of your public key that you generated. Send this email to [adele@gnupp.de](mailto:adele@gnupp.de) from which you should receive a return email from Adele that will have Adele's public key, which you will have to process using Kleopatra.

The Adele robot will return its public key, which you should save to the same directory from which you exported and stored information about your generated key. Name the file something like AdelePublicKey.asc. You can now import the public key returned by Adele by cutting and pasting your email or from the file that we just created, and associate it with the [adele@gnupp.de](mailto:adele@gnupp.de) email address. The Gpg4win manual section titled "Importing a public certificate" describes in detail how to do this. ***Save Adele's public key to the file AdelePublicKey.asc > in Kleopatra click the Import Certificates… button upper left > navigate to the file containing the public key and select it, Open > the Detailed results of importing C:\...\AdelePublicKey.asc dialog should confirm that the certificate was processed and imported***. The text you want to cut and paste into the file is the following:

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v1.4.10 (GNU/Linux)

All text down to the following…

-----END PGP PUBLIC KEY BLOCK-----

In the public key message returned by Adele, there is a message in German that when translated to English using <http://translate.google.com> states:

This is more than is normally the case for testing. Since can email addresses easily fake and I by a stranger an excessive number of emails will bother, I would like to take this opportunity make sure the address is correct. To make this confirm, please send me a reply that the string `AD-DYZELXHIPLFEAXXA 'in the subject line contains. In a normal response to this email, this should automatically happen.

ATTENTION! As long as I do not receive this confirmation, will NO I send more emails to the specified address. As soon as I get the confirmation, you can think continue to use service. More confirmations are then every 25 emails due.

While this translation is not perfect, we get the gist that after you get Adele's public key response you must reply with the string above in your email subject line before you can begin exchanging encrypted email with the Adele robot. If you do not do this, you will get a response from [adele-abuse@gnupp.de](mailto:adele-abuse@gnupp.de) stating:

The following email is rejected because the correctness of the address given has not been confirmed yet.

The Gpg4win manual describes in detail how to encrypt your email in Outlook using their GpgOL utility. It is as simple as selecting ***Extras > Encrypt message*** in the menu of the message window. The lock button will appear stating encrypt message with GnuPG and then all you have to do is send the message. Gpg4win will automatically detect the OpenPGP protocol and the public key of your partner that you imported to Kleopatra.

To do this using Claws, Thunderbird or some other email client is somewhat more complex since we don't have the GpgOL utility to automate this task for us. In Claws, the first step is to load the PGP/Core, PGP/Inline and PGP/MIME plugins. The PGP/Core plugin provides the core components of the Claws Mail PGP system. The PGP/MIME plugin allows you to send encrypted messages as attachments and the PGP/Inline handles signed and/or encrypted email. ***Click on the Configuration menu > at the bottom select Plugins… > in the Plugins dialog click on the Load… button > select pgpcore.dll, Open > Load… select pgpinline.dll, Open > Load… select pgpmime.dll, Open > Close***. Once the needed plugins are loaded we can now use the Configuration menu to setup our encryption preferences. ***Click on Configuration > arrow down to select Preferences… > and decide which options you may want, such as checking Automatically check signatures***.

We are now ready to send an encrypted message to [adele@gnupp.de](mailto:adele@gnupp.de) by ***clicking on the Options menu > arrow down to select Privacy System and select the encryption method you plan to use:***

* ***None: Use no encryption***
* ***PGP Inline: Encrypt your message inline***
* ***PGP MIMI: Encrypt your message and add it as an attachment***

***Click on the Options menu again > arrow down to select Encrypt and consider also selecting Sign if you have certified/verified the public key in Kleopatra > click on Send to encrypt and send the message to the public key of your choosing, if you have more than one > you may see an Encryption warning dialog warning you that attachments and headers are not encrypted, Continue > enter your passphrase and the encrypted message will be sent***. To make sure everything worked ***click on the Get Mail button upper left > select the Claws Sent folder and you should see a key next to the encrypted email that you just sent > view the message by double clicking on it, and only for the first time will you have to enter your passphrase***. This proves that the message was encrypted without having to call or experiment upon one of your partners. Adele will respond with an automatic unencrypted email showing that they received and decrypted your email. To prove that Adele received and decrypted your email, send them a decrypted email and look at the reply. It will just contain their encrypted pubic key showing that their automated server did not understand your unencrypted message.

Once you have practiced these steps and are confident in their use, you need to change your digital life and commit to using email message encryption technology. ***When you generate your certificate for real using Kleopatra, click on File > New Certificate… > on the Choose Certificate Format dialog, click on Create a personal OpenPGP key pair, Next > on the Enter Details screen, click on the Advanced Setting… button lower right and under Key Material increase the RSA key length to 4096, which is the MAX currently offered by Kleopatra > check the Authentication box so that you can include the user IDs of other certificate holders who have confirmed the authenticity of your certificate, OK > Enter your Name:, EMail:, and Comment: details. By selecting the Authentication box you can establish a "Web of Trust" which will save your partner's time and effort authenticating your public key certificate as you share it. Consider setting an expiration date as there is no need to have these keys hanging around on the Internet forever.*** Setting a lengthy expiration date is a courtesy to key servers' administrators and to your partners as it shows them when obsolete keys can be removed and possibly the need to be renewed at higher levels of encryption. For example, eight years is more than long enough for a key pair to exist as technology and encryption techniques advance. Remember, authentications are only relevant to OpenPGP certificates. Modern computers can process a 4096 key length with minimal effort for your added security. In eight years you may want 8192 or greater.

You also have to take a moment to plan where your keys will be stored, how and where you are going to back them up, and how you are going to use them. For example, if you are an SBO, you may want just one set of keys available to everyone that uses your computer(s) to allow the exchange of encrypted messages with your partners. However, if you are doing this just for yourself, then the defaults may be fine.

Section 18 of the Gpg4win describes very well how to sign and encrypt a file. Once again, sending an encrypting a file is very simple using GpgEX. ***Using Windows Explorer browse to the files or folders that you want to select and right click on the mouse to bring up the menu > arrow down to select Sign and encrypt > on the Sign/Encrypt Files dialog choose what you want to do***. From <http://www.gpg4win.org/doc/en/gpg4win-compendium_24.html>:

* You sign a file using your private certificate to ensure that the file cannot be modified.
* Then encrypt the file using a public certificate to prevent unauthorized persons from seeing it.

Please note these lessons are intended for email clients such as Claws that do not have GpgOL that will automatically sign and encrypt your attachments, which you can easily do in Windows Outlook. There are plenty of simple web pages on the Internet that will show you how to do this in Outlook.

The lessons above were meant to get you started and to show how easy it is to put your email in an encrypted envelope. It is up to you to take these lessons to heart, learn more and apply encryption to your digital use of email. I hope you can see how easy all of this is to protect your digital privacy. An example of a public key server is <http://www.rossde.com/PGP/pgp_keyserv.html> by David Ross--PGP Public Key Servers--Mozilla Firefox.

### Thunderbird -- Using Enigmail to Encrypt Emails

Enigmail is a plug-in for the Thunderbird or Seamonkey email clients that allow them to interface easily with GnuPG. Mozilla Thunderbird is a versatile email client that is available for GNU Linux, Mac OS, Microsoft Windows and other OSs. To use Enigmail you will need to download the latest version of Thunderbird from their web page <https://www.mozilla.org/en-US/thunderbird> and install it. Once installed, you will have to configure Thunderbird to work with your email service providers. Thunderbird is very good at figuring out how to configure itself for most email service providers, so most likely all you will have to do is enter your username and password. Once you have set up your email accounts in Thunderbird, refer back to [Windows 7-Using Gpg4win to Encrypt Emails & Files](#_Windows_7-Using_Gpg4win) to install their project's encryption software. Once installed, we need to add the Enigmail add-on to be able to encrypt and decrypt email. To install the Enigmail add-on:

1. ***At the top of Thunderbird right click on top bar and check Menu Bar to add the Menu Bar***.
2. ***In Mozilla Thunderbird, click on the Tools menu > arrow down to select Add-ons***.
3. ***In the search box upper right type enigmail and you should see Enigmail 1.6 > click on the Install button on the right to install the add-on***.
4. ***If your search does not find Enigmail download surf to*** [*https://addons.mozilla.org/en-US/thunderbird/addon/enigmail*](https://addons.mozilla.org/en-US/thunderbird/addon/enigmail) ***or*** [*https://www.enigmail.net/home/index.php*](https://www.enigmail.net/home/index.php) ***and download the add-on which will work in Thunderbird > save the enigmail-\*.xpi file to your hard drive.***
5. ***From the option button next to the top-right add-on search field, select Install Add-on From File… and browse to the downloaded add-on file to install it***.

In the Gpg4win section of this chapter above we generated a John Doe key pair to practice using with the Adele robot using the Claws email client. We will again use that same key pair to practice encrypting email using Thunderbird/Enigmail before we venture out into the real world. Enigmail comes with a setup wizard that you can run any time by ***clicking on the Thunderbird OpenPGP menu added when you installed the add-on > arrow down to select Setup Wizard > the Welcome to OpenPGP Setup Wizard should appear, leave Yes, I would like the wizard to get me started ticked, Next > > on the Select Identities dialog select the email accounts with which you plan to use to encrypt email, Next > > on the Digitally Sign Your Outgoing Emails screen tick No, I want to create per-recipient rules for emails that need to be signed, Next > > on the Encrypt Your Outgoing Emails screen tick No, I will create per-recipient rules for those that sent me the public key, Next > > on the Change Your Email Settings To Make OpenPGP Work More Reliably screen, allow Enigmail to change your defaults settings, leave Yes ticked, Next > > on the Create A Key To Sign And Encrypt Email select the John Doe (Test) key we created while studying Gpg4win, Next > > on the Summary screen confirm your choices, Next > and we are ready to practice sending encrypted emails using Enigmail***. (See also: <https://securityinabox.org/en/thunderbird_main>)

|  |  |
| --- | --- |
| Tip | Most people you send email to care nothing about their privacy and drag you down that rabbit hole. They do not see the need to set up an OpenPGP-aware email program, and your signature will be displayed as an attachment or as text around the email messages you send. This will annoy and confuse them resulting in endless questions and explanations from yourself. In a perfect world, leaving Yes, I want to sign all my email ticked would be appropriate, and to know why, refer back to [Digital Signatures and Why they are Important](#_Digital_Signatures_&) to understand their purpose. The same holds true with the next Encrypt Your Outgoing Emails screen where you can choose to encrypt all your email. Accidentally send one encrypted email to someone who knows nothing about encryption, and you will get grief about how you are sending gibberish, and most likely an accusation from them about how you must have a virus on your computer. Maintaining a separate email address for all of your encrypted communication is a wonderful solution that you should consider. |

In the above settings when asked if we wanted to change a few default settings to make OpenPGP work better, we answered "Yes". By doing this the following changes were made to our Thunderbird configuration:

* Disable loading IMAP parts on demand
* Disable flowed text (RFC 2646)
* View message body as plain text (Enigmail does not work well with HTML)
* Use 8-bit encoding for message sending
* Do not compose HTML messages

The first step to test and learn about Enigmail is to sign a message and send it to Adele. This will allow Enigmail to sign your email and send it off to its lists. Only sign the email and do not encrypt it. ***Click on the Inbox of the email account you are using > click on the Write icon to draft a message and put together a message to send to Adele > click on OpenPGP menu at the very top and arrow down to select Sign Message > then click on Send to deliver it to Adele***. Adele should respond with its public key, which you can import if you have not already done so. The next step is to send an encrypted email to Adele.

*Click on the Inbox of the email account you are planning to use > click on the Write icon to draft a message and put together an email message to send to Adele > click on OpenPGP and arrow down to check Encrypt Message > you may be prompted to configure OpenPGP for this email identity click on Configure > check Enable OpenPGP support (Enigmail) for this identity > click on the Select Key… button and you may want to select your John Doe key that we created while learning about Gpg4win, OK > click on the Send button, enter your passphrase and wait for a response from Adele*. It is not necessary to select your own key unless you want to review your sent encrypted emails later, as an SBO may want to do. What Adele sends back will be decrypted if everything is working OK. If the response from Adele comes back in German unencrypted with something like the following:

Hallo ThatCyberSecurityGuy,

hier ist die unverschlüsselte Antwort auf Ihre EMail.

Ich habe von Ihnen eine verschlüsselte Botschaft empfangen.

Allerdings finde ich keinen öffentlichen Schlüssel von Ihnen, sodass

ich Ihnen nicht verschlüsselt antworten kann. Da man verschlüsselte

Texte vertraulich behandeln, also nicht unverschlüsselt

weiterschicken soll, gehe ich in dieser Antwort nicht auf den Inhalt

Ihrer verschlüsselten EMail ein, sondern sage Ihnen hiermit nur, dass

ich sie entschlüsseln konnte.

Go to <http://translate.google.com> and you will see that Adele is telling you it received an encrypted email but could not find a public key from you. Open up Kleopatra to make sure Adele's public key has been imported and registered. OpenPGP should recognize the [adele@gnupp.de](mailto:adele@gnupp.de) email address and associate it with the proper public key when you send your email. *Click on OpenPGP > arrow down to select Default composition options > arrow over to select Signing/Encryption Options… > make sure Use email address of this identity to identify OpenPGP key is selected. If that does not work, send your John Doe public key to Adele again to get the robot to reregister it*. Adele does not keep your public key long for obvious reasons, and you have to confirm your pubic key with the string Adele sent you in its public key every 25 emails. When you get an email response from Adele looking like the following, you have mastered the novice level of using email encryption:

\*\*\*\*\*\*\*\*\* \*BEGIN ENCRYPTED or SIGNED PART\* \*\*\*\*\*\*\*\*\*

Hallo ThatCyberSecurityGuy,

hier ist die verschlüsselte Antwort auf Ihre EMail.

Ich schicke Ihnen Ihre Botschaft im Wortlaut zurück, damit Sie

sehen, dass ich sie erfolgreich entschlüsseln konnte.

> Test encrypted email to Adele public key...

Viele Grüße,

adele@gnupp.de

\*\*\*\*\*\*\*\*\*\* \*END ENCRYPTED or SIGNED PART\* \*\*\*\*\*\*\*\*\*\*

Which, when translated to English, means:

Here is the encrypted response to your email. I'll send you back your message in the text, so you see that I was able to decode successfully.