Project 1c. Use gpg to sign and verify open source software.

The learning goals of this assignment is to

- learn how to create your own public/private key using gpg.
- publish your own public key on gpg server and your AWS web server.
- retrieve a public key from a gpg server and import into your key ring.
- learn how to sign a software and post the the signature/software on a web server
- learn how to verify the open source software with PGP signature.

The assignment will be graded by

1. whether the public key is properly posted in the learner's web server
2. whether the software is properly signed, and the signature and software are properly posted for others to verify
3. whether the learner has demonstrated proper verification of an opensource software package.

Assignment Topic:

Use gpg to generate your public/private keys, publish your public key, download and verify the authenticity and integrity of open source software packages, sign a software package and post them properly.

Setup instructions:

Before you begin, you will need to login to your instance following the steps in Project 1a. Create a pgp directory using "mkdir pgp". Switch to the pgp directory with "cd pgp".

Step 1. Generate the public/private key pair and publish the public key.

Run the command “gpg --gen-key”

gpg --gen-key

It will show the version of gpg software that is installed in your instance.

If this is the first time you run the command, it will create a .pgp in your home directory and create public keyring and private keyring there to save the created or imported keys.

It will then ask you to specify the type of key you like to create. We will select 1 which is RSA and RSA algorithm. We will choose 4096 bit as key size and choose it to be valid forever by entering 0.

Gpg then asks you to enter the USER_ID information. Enter your full name as Real Name, email address you used in your Coursera account as Email address, and "Test Account for Coursera CS5910" as Comment. Enter O for "Okay".

Gpg then ask you to enter the passphrase to protect your private key. Make sure you remember it.

The following is the session data shown when we execute this command. Make sure to replace with your own User_ID info.

TIPS: Save the sessions of your project as text files. They contain rich information spit out by the gpg command. You can review them offline and learn from them.

```
gpg (GnuPG) 2.0.28; Copyright (C) 2015 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.
gpg: directory `/home/ec2-user/.gnupg' created
gpg: new configuration file `/home/ec2-user/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/ec2-user/.gnupg/gpg.conf' are not yet active
during this run
gpg: keyring `/home/ec2-user/.gnupg/secring.gpg' created
gpg: keyring `/home/ec2-user/.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) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: C. Edward Chow
Email address: edwardchowc@gmail.com
Comment: Test Account for Coursera CS5910
You selected this USER-ID:
    "C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
```

You got the prompt asking you to enter the passphrase to protect your public and private keys.

You will asked again later to enter passphrase when encrypt files.

Step 2. Publish your public key.
You can

1) publish your public key on your web server as a web page.
2) Attach your public key as attachment or main text in your email. The recipient can import them to their public key ring.
3) Publish your public key to pgp key servers.

To publish public key as a web page or text email attachment, first we need to extract the public key and output them in ASCII readable format. We run the following command:

```
gpg --armor --export edwardchowc@gmail.com > mypubkey.asc
```

--armor option generate ASCII instead of binary file.
--export option extract the related public key out from the public keyring. We can use KeyID (portion of key fingerprint) or email address.
We redirect the result on console to save as a file with .asc (which is file extension for GPG public key in ASCII format; the binary file with have .pgp as file extension).

Let us create a pgp directory on our web site and copy the mypuykey.asc there.

```
mkdir /var/www/html/pgp;  cp mypubkey.asc /var/www/html/pgp
```

If you see permission denied message, add "sudo" before your command.
Your ec2-user account may have the write privilege on /var/www/html which is typically owned by root. You can verify the web access by typing the following url into your browser.
https://<yourInstanceIP>/pgp/mypubkey.asc
You can now tell your friend to retrieve your public key if you add their local IP address to the instance's source IP address list. For this project, you should enter 128.198.0.0/16 into your instance's source IP addresss list.

Capture the image showing on the browser of your public key content.
The image is the first deliverable of the project1c you should submit.

To publish the public key to a pgp key server, run the following command:

```
gpg --keyserver pgp.mit.edu --send-key edawrdchowc@gmail.com
```

Note that the pgp key servers are synchronized, once key is published, it will be soon available on all pgp key servers.

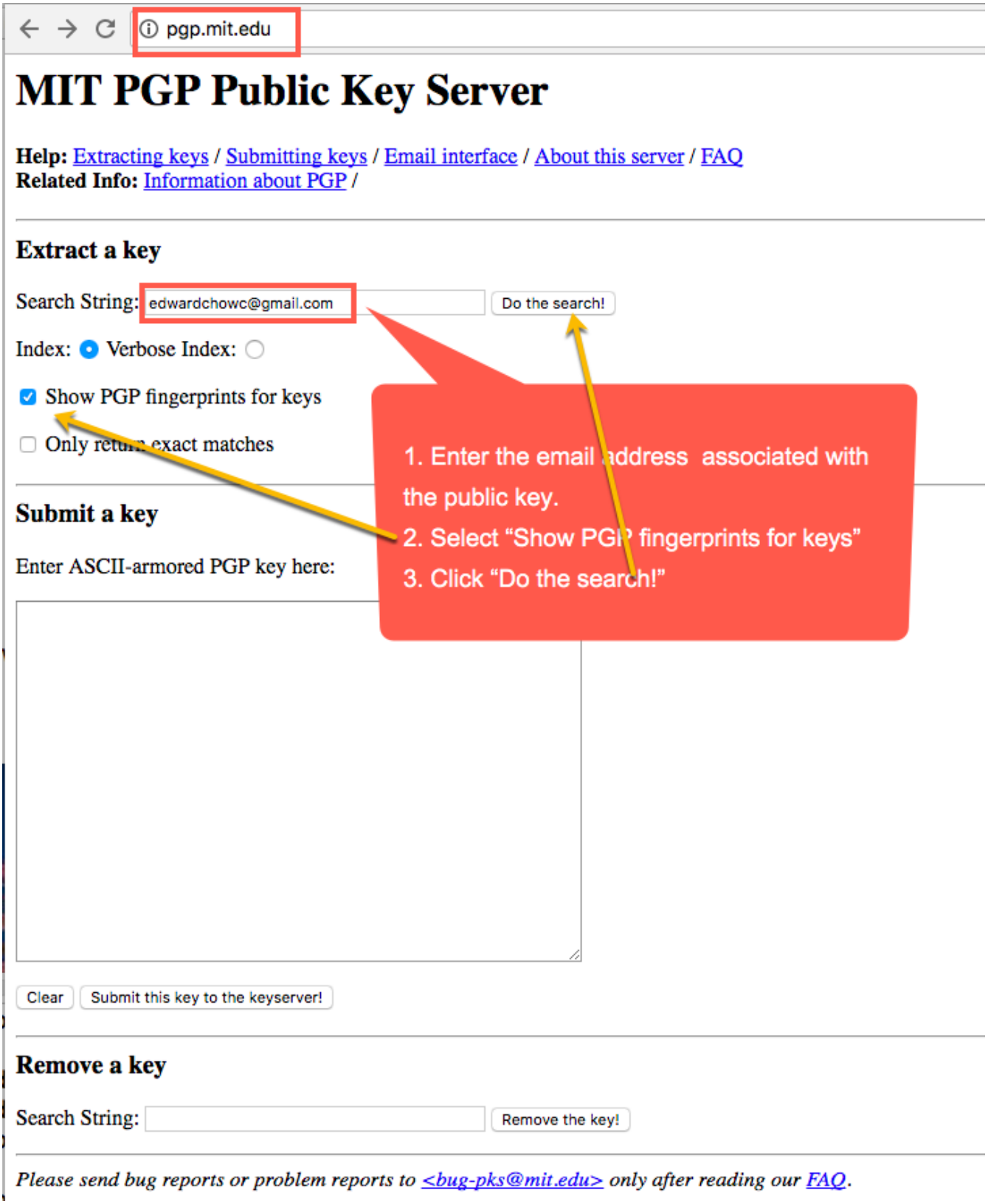
To retrieve a public key, you can use the gpg command with --search-key option. The gpg can search for the key server preconfigured or you can specify a specific keyserver you trust or close-by.

gpg [--keyserver pgp.mit.edu] --search-keys <keyID or email address>

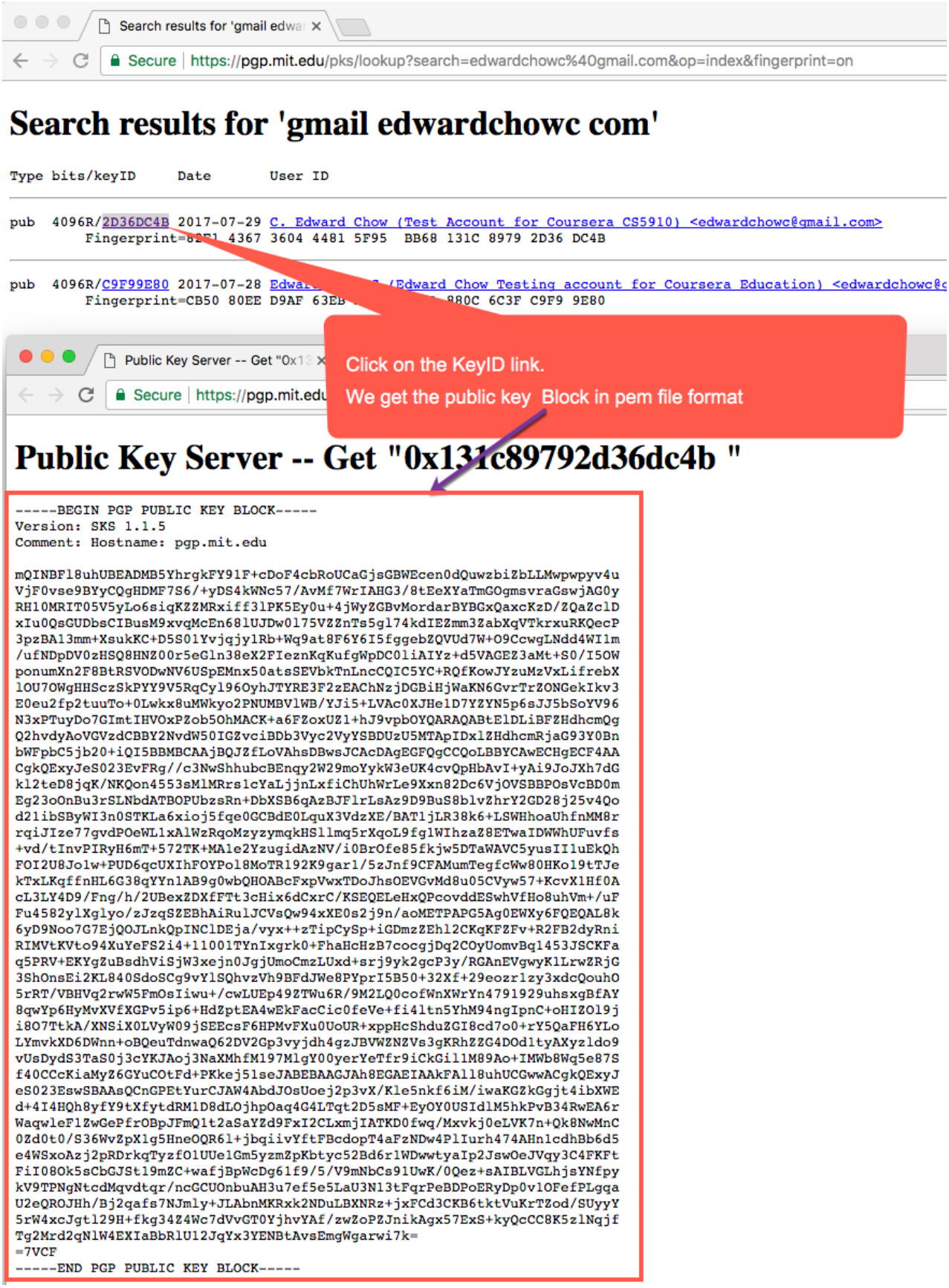
```
$ gpg --keyserver pgp.mit.edu --search-keys edwardchowc@gmail.com
gpg: searching for "edwardchowc@gmail.com" from hkp server pgp.mit.edu
(1)   C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.c
        4096 bit RSA key 2D36DC4B, created: 2017-07-29
(2)   Edward Chow C (Edward Chow Testing account for Coursera Education) <ed
        4096 bit RSA key C9F99E80, created: 2017-07-28
Keys 1-2 of 2 for "edwardchowc@gmail.com".  Enter number(s), N)ext, or Q)uit > 1
gpg: requesting key 2D36DC4B from hkp server pgp.mit.edu
gpg: key 2D36DC4B: "C. Edward Chow (Test Account for Coursera CS5910)
<edwardchowc@gmail.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
```

You can also enter https://pgp.mit.edu/ or any pgp key server with GUI. Enter search string and get the public key on the web browser. You can then copy and save the pub key as .pem file and import into your key ring.

Here are steps:



and the search results:


Search results for 'gmail edwa
Secure | https://pgp.mit.edu/pks/lookup?search=edwardchowc%40gmail.com&op=index&fingerprint=on
Search results for 'gmail edwardchowc com'
Type bits/keyID Date User ID
pub 4096R/2D36DC4B 2017-07-29 C. Edward Chow (Test Account for Coursera CS5910) <edwardchowc@gmail.com>
Fingerprint=82E1 4367 3604 4481 5F95 BB68 131C 8979 2D36 DC4B
pub 4096R/C9F99E80 2017-07-28 Edwa (Edward Chow Testing account for Coursera Education) <edwardchowc@
Fingerprint=CB50 80EE D9AF 63EB 980C 6C3F C9F9 9E80
Public Key Server -- Get "0x13
Secure | https://pgp.mit.edu
Click on the KeyID link.
We get the public key Block in pem file format
Public Key Server -- Get "0x131c89792d36dc4b "
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: SKS 1.1.5
Comment: Hostname: pgp.mit.edu
mQINBFl8uhUBEADMB5YhrgkFY91F+cDoF4cbRoUCaGjsGBWEcen0dQuwzbiZbLLMwpwpyv4u
VjF0vse9BYyCQgHDMF7S6/+yDS4kWNc57/AvMf7WrIAHG3/8tEeXYaTmGOgmsvraGswjAG0y
RH10MRIT05V5yLo6siqKZZMRxiff3lPK5Ey0u+4jWyZGBvMordarBYBGxQaxcKzD/ZQaZc1D
xIu0QsGUDbsCIBusM9xvqMcEn68lUJDw0175VZZnTs5gl74kdIEZmm3ZabXqVTkrxuRKQecP
3pzBAl3mm+XsukKC+D5S01Yvjqjyl Rb+Wq9at8F6Y6I5fggebZQVUd7W+O9CcwgLNdd4WI1m
/ufNDpDV0zHSQ8HNZ00r5eGln38eX2FIeznKqKufgWpDC0liAIYz+d5VAGEZ3aMt+S0/I5OW
ponumXn2F8BtRSVODwNV6USpEMnx50atsSEVbkTnLncCQIC5YC+RQfKowJYzuMzVxLifrebX
lOU7OWgHHSczSkPYY9V5RqCyl96OyhJTYRE3F2zEAChNzjDGBiHjWaKN6GvrTrZONGekIkv3
E0eu2fp2tuuTo+0Lwkx8uMWkyo2PNUMBVlWB/YJi5+LVAc0XJHe1D7YZYN5p6sJJ5bSoYV96
N3xPTuyDo7GImtIHVOxPZob5OhMACK+a6FZoxUZ1+hJ9vpbOYQARAQABtElDLiBFZHdhcmQg
Q2hvdyAoVGVzdCBBY2NvdW50IGZvciBDb3Vyc2VyYSBDUzU5MTApIDxlZHdhcmRjaG93Y0Bn
bWFpbC5jb20+iQI5BBMBCAAjBQJZfLoVAhsDBwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AA
CgkQExyJeS023EvFRg//c3NwShhubcBEnqy2W29moYykW3eUK4cvQpHbAvI+yAi9JoJXh7dG
kl2teD8jqK/NKQon4553sMlMRrs1cYaLjjnLxfiChUhWrLe9Xxn82Dc6VjOVSBBPOsVcBD0m
Eg23oOnBu3rSLNbdATBOPUbzsRn+DbXSB6qAzBJFlrLsAz9D9BuS8blvZhrY2GD28j25v4Qo
d21ibSByWI3n0STKLa6xioj5fqe0GCBdE0LquX3VdzXE/BAT1jLR38k6+LSWHhoaUhfnMM8r
rqiJIze77gvdPOeWL1xAlWzRqoMzyzymqkHSllmq5rXqoL9fg1WIhzaZ8ETwaIDWWhUFuvfs
+vd/tInvPIRyH6mT+572TK+MAle2YzugidAzNV/i0BrOfe85fkjw5DTaWAVC5yusII1uEkQh
FOI2U8Jo1w+PUD6qcUXIhFOYPol8MoTR192K9gar1/5zJnf9CFAMumTegfcWw80HKo19tTJe
kTxLKqffnHL6G38qYYn1AB9g0wbQHOABcFxpVwxTDoJhsOEVGvMd8u05CVyw57+KcvXlHf0A
cL3LY4D9/Fng/h/2UBexZDXfFTt3cHix6dCxrC/KSEQELeHxQPcovddESwhVfHo8uhVm+/uF
Fu4582ylXglyo/zJzqSZEBhAiRulJCVsQw94xXE0s2j9n/aoMETPAPG5Ag0EWXy6FQEQAL8k
6yD9Noo7G7EjQOJLnkQpINClDEja/vyx++zTipCySp+iGDmzZEhl2CKqKFZFv+R2FB2dyRni
RIMVtKVto94XuYeFS2i4+l1001TYnIxgrk0+FhaHcHzB7cocgjDq2COyUomvBq1453JSCKFa
q5PRV+EKYgZuBsdhViSjW3xejn0JgjUmoCmzLUxd+srj9yk2gcP3y/RGAnEVgwyKlLrwZRjG
3ShOnsEi2KL840SdoSCg9vYlSQhvzVh9BFdJWe8PYprI5B50+32Xf+29eozrlzy3xdcQouhO
5rRT/VBHVq2rwW5FmOsIiwu+/cwLUEp49ZTWu6R/9M2LQ0cofWnXWrYn4791929uhsxgBfAY
8qwYp6HyMvXVfXGPv5ip6+HdZptEA4wEkFacCic0feVe+fi4ltn5YhM94ngIpnC+oHIZOl9j
i8O7TtkA/XNSiX0LVyW09jSEEcsF6HPMvFXu0UoUR+xppHcShduZGI8cd7o0+rY5QaFH6YLo
LYmvkXD6DWnn+oBQeuTdnwaQ62DV2Gp3vyjdh4gzJBVWZNZVs3gKRhZZG4DOd1tyAXyzldo9
vUsDydS3TaS0j3cYKJAoj3NaXMhfM197MlgY00yerYeTfr9iCkGil1M89Ao+IMWb8Wq5e87S
f40CCcKiaMyZ6GYuCOtFd+PKkej51seJABEBAAGJAh8EGAEIAAkFAll8uhUCGwwACgkQExyJ
eS023EswSBAAsQCnGPEtYurCJAW4AbdJOsUoej2p3vX/Kle5nkf6iM/iwaKGZkGgjt4ibXWE
d+4I4HQh8yfY9tXfytdRM1D8dLOjhpOaq4G4LTqt2D5sMF+EyOY0USIdlM5hkPvB34RwEA6r
WaqwleF1ZwGePfrOBpJFmQ1t2aSaYZd9FxI2CLxmjIATKD0fwq/Mxvkj0eLVK7n+Qk8NwMnC
0Zd0t0/S36WvZpX1g5HneOQR6l+jbqiivYftFBcdopT4aFzNDw4PlIurh474AHn1cdhBb6d5
e4WSxoAzj2pRDrkqTyzfOlUUe1Gm5yzmZpKbtyc52Bd6r1WDwwtyaIp2JswOeJVqy3C4FKFt
FiI08Ok5sCbGJSt19mZC+wafjBpWcDg61f9/5/V9mNbCs91UwK/0Qez+sAIBLVGLhjsYNfpy
kV9TPNgNtcdMqvdtqr/ncGCUOnbuAH3u7ef5e5LaU3N13tFqrPeBDPoERyDp0v1OFefPLgqa
U2eQROJHh/Bj2qafs7NJmly+JLAbnMKRxk2NDuLBXNRz+jxFCd3CKB6tktVuKrTZod/SUyyY
5rW4xcJgtl29H+fkg34Z4Wc7dVvGT0YjhvYAf/zwZoPZJnikAgx57ExS+kyQcCC8K5zlNqjf
Tg2Mrd2qNlW4EXIaBbRlUl2JqYx3YENBtAvsEmgWgarwi7k=
=7VCF
-----END PGP PUBLIC KEY BLOCK-----

Compare the https://<yourInstanceIP>/pgp/mypubkey.asc content with the public key pgp.mit.edu key server displayed and see if they are different. Capture the image of mypubkey.asc on your browser and save it as mypubkey.png and capture the image of your publick key on pgp.mit.edu or any public key server. Create a comparePubKey.html in /var/www/html/pgp of your instance and include the brief comparison results of mypubkey.asc vs the public key displayed by pgp.mit.edu, and include those two images you just captured.

Submit the two images and the comparison as your first deliverable of the project 1c.

Just first 10 and last 10 characters are enough to tell the differences. Now you are sure that the public key exported out from your local keyring is the same as those posted on the key server. Note that the versions header generated by your gpg command and that of pgp.mit.edu are different. One has value of GnuPG v2; the other has SKS 1.1.5.

Step 3. Verify Opensource Software Packages.

We will use curl commands to download the source code of apache httpd source code and related key files. The – O option save the file with the same file name of the url. Then use gpg --verify to verify the authenticity and integrity of the download source. Apache has a lot of projects. All their code signing public keys are consolidated in a single KEYS file and make it easier for the user to retrieve. They should include the link to this KEYS file in the specific software download pages.

```
$ curl -O http://archive.apache.org/dist/httpd/KEYS
$ curl -O http://archive.apache.org/dist/httpd/httpd-2.4.25.tar.bz2
$ curl -O http://archive.apache.org/dist/httpd/httpd-2.4.25.tar.bz2.asc

$ gpg --import KEYS
$ gpg --verify httpd-2.4.25.tar.bz2.asc httpd-2.4.25.tar.bz2
```

putty keys are described here
https://www.chiark.greenend.org.uk/~sgtatham/putty/keys.html

```
$ curl -O https://the.earth.li/~sgtatham/putty/0.70/w64/putty-64bit-0.70-
   installer.msi
$ curl -O https://the.earth.li/~sgtatham/putty/0.70/w64/putty-64bit-0.70-
   installer.msi.gpg
$ curl -O https://www.chiark.greenend.org.uk/~sgtatham/putty/keys/release-2015.asc
$ gpg --import release-2015.asc
gpg: key B43434E4: public key "PuTTY Releases <putty@projects.tartarus.org>"
imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 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 --verify putty-64bit-0.70-installer.msi.gpg putty-64bit-0.70-installer.msi
gpg: Signature made Sat 08 Jul 2017 06:49:50 AM UTC using RSA key ID B43434E4
gpg: Good signature from "PuTTY Releases <putty@projects.tartarus.org>" [unknown]
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 0054 DDAA 8ADA 15D2 768A  6DE7 9DFE 2648 B434 34E4
```

Save the above gpg –import and gpg –verify execution results as a web page called puttyVerified.html by wrapping the text session data with <pre></pre> tag.

```
<h1>putty software verification results</h1>
<pre>

--- put the putty gpg verification text session data here ---

</pre>
```

copy it to your pgp web site with the following command.

```
cp  puttyVerified.html  /var/www/html/pgp/
```

Capture the image of http://<yourInstancePublicIP>/pgp/puttyVerified.html and submit it as the third deliverable of your project 1c.

Note that if you retrieve the older putty release key with link in E.3 section of https://www.chiark.greenend.org.uk/~sgtatham/putty/keys.html

Then the key will be rejected due to the use of older MD5 algorithm.

```
gpg --import release-rsa.asc
gpg: Note: signatures using the MD5 algorithm are rejected
gpg: key B41CAE29: no valid user IDs
gpg: this may be caused by a missing self-signature
gpg: Total number processed: 1
gpg:           w/o user IDs: 1
```

Step 4. Sign a software with your private key and post it properly.

Rename the httpd-2.4.25.tar.bz2 in your pgp directory as httpd.bz2

```
cp httpd-2.4.25.tar.bz2  httpd.bz2

gpg --output httpd.bz2.gpg --detach-sig httpd.bz2

cp httpd.bz2 httpd.bz2.gpg  /var/www/html/pgp
```

Create a software download page called myhttpd.html with the following content

```
<h1> my signed httpd software package</h1>
```

```
My software <a href="httpd.bz2">httpd.bz2</a> its signature <a
href="httpd.bz2.gpg">signature</a> and my software release public key <a
href="mypubkey.asc">.

---Add a description on how gpg can be used to verify your httpd.bz2 software.---

---Add the gpg session with httpd.bz2.gpg is indeed signed by you  ---

cp myhttpd.html to /var/www/html/pgp
```

Capture the image of your browser displaying http://<yourInstancePublicIPAddress>/pgp/myhttpd.html as your 4th deliverable of your Project 1c.

You grade will be based on how concise and clear your description on how to verify your httpd.bz2 with proper gpg commands.

How to submit:

You will be required to submit your assignment with the images of

- https://<yourInstancePublicIP>/pgp/comparePubKey.html
- https://<yourInstancePublicIP>/pgp/puttyVerified.html
- https://<yourInstancePublicIPAddress>/pgp/myhttpd.html

How to create your assignment:

Follow the instructions in the project description to set up those web page answers.

Prompt 1.
Does the user properly post the public key on its web site?
Does the comparison (2nd url) clearly indicates the public key on local key ring are the same as those on the key server?

Prompt 2.
Does the puttyVerified.html web page clearly shows the learner has verified that the putty software is from the original source and its integrity is not violated?

Prompt 3.
Does the learner properly display all information necessary for others to download and verify the httpd.bz2.gpg posted on the web site?