

Gnu Privacy Guard

Introduction

Goals

- For LA County agencies, distributed applications and data sharing has expanded beyond agency boundaries. We share data between an agency and
 - another government agency,
 - an external consultant or contractor,
 - the support staff of a vendor.
- The context of this presentation includes:
 - Sharing small amounts of sensitive information such as API credentials between the maintenance staff of client and service endpoint providers.
 - Transmitting application logs with sensitive information.
 - Transmission is manual, usually via email.
 - This is **not** a presentation about application-to-application security.

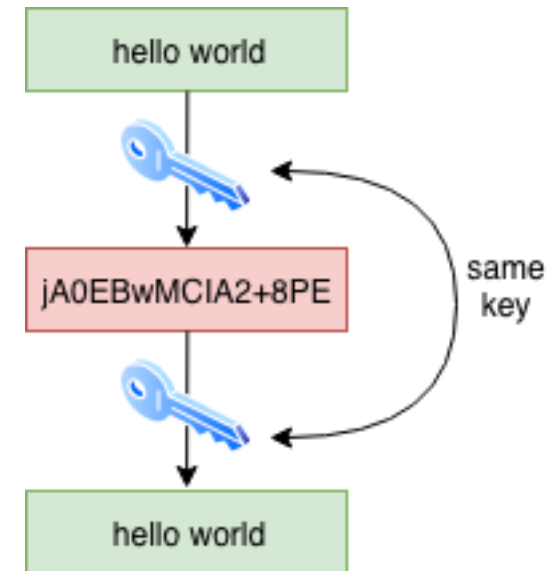
Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing

- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

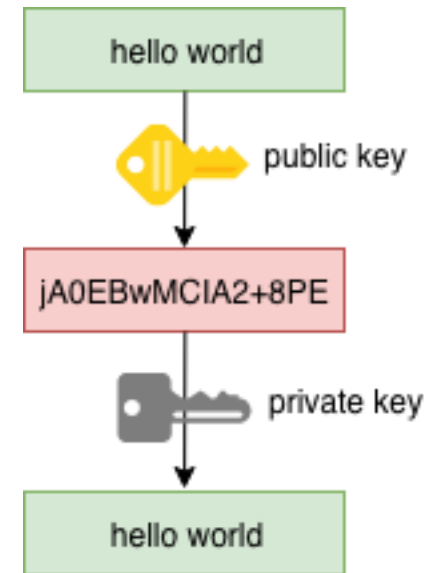
Concept Review – Symmetric Encryption

- A single encryption key encrypts and decrypts a message.
- Similar in concept to a physical key that can lock and unlock a door.
- Pros
 - Conceptually simple.
 - Computationally efficient.
- Cons
 - When transmitting encrypted information, there is the problem of transmitting the key.
 - This is usually not a problem between two individuals who already know each other.
 - It does not scale as the number of parties increases.



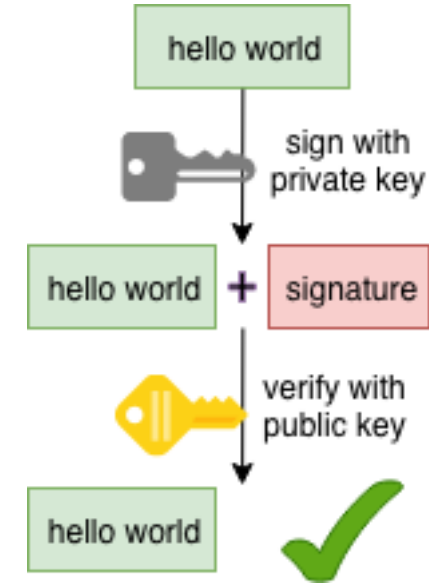
Concept Review – Asymmetric Encryption

- Uses a key-pair.
- One key encrypts, the other decrypts.
 - may start with either key
 - but only the other key will decrypt
 - conventionally assigned as public key and private key.
 - anyone with a public key can encrypt; only the private key holder can decrypt.
- Pros
 - Public key not compromised by disclosure.
- Cons
 - Computationally inefficient.



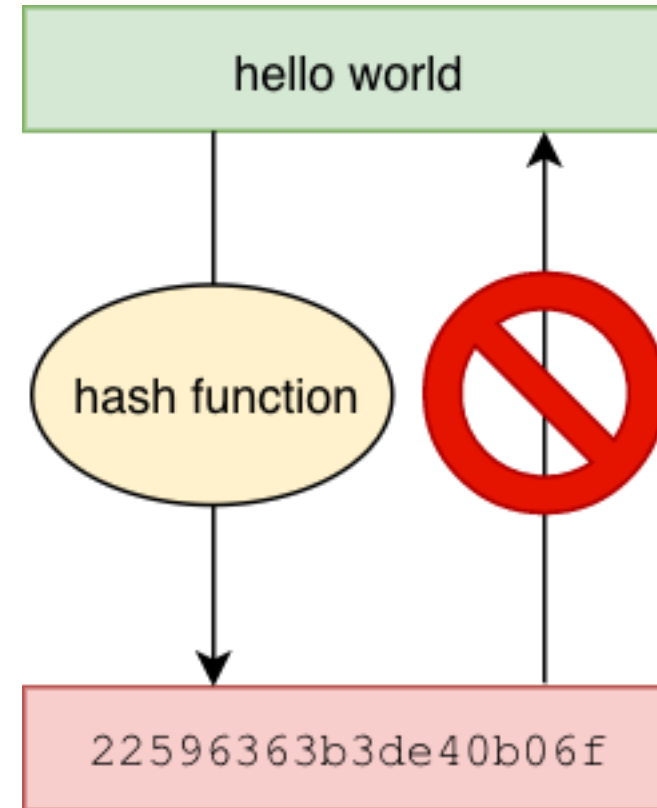
Concept Review - Digital Signature

- Signatures assure the receiver
 - who the sender was, and
 - contents were not altered in transit.
- Independent of encryption.
- Conceptually, it's asymmetric encryption in reverse.
 - only the sender can sign.
 - anyone (with the public key) can verify.



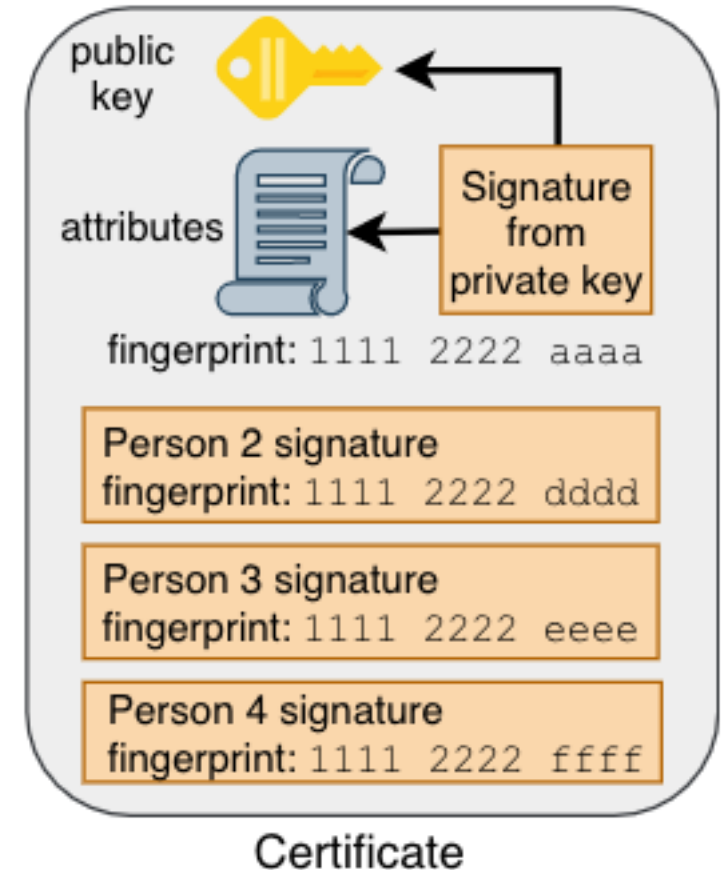
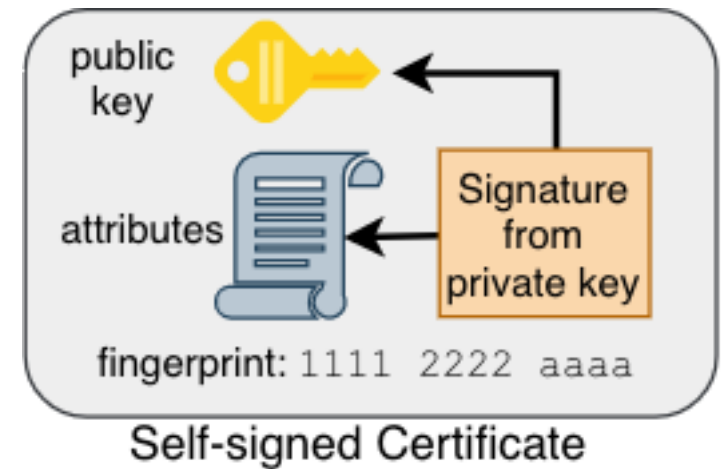
Concept Review - Hash

- A hashing function take any data as input and produces a fixed length result in a repeatable way.
- The term hash is a
 - **noun** – the output of the function
 - **verb** – the act of applying the function
- Qualities of a “good” hash function
 - difficult to reverse
 - produces very different outputs for inputs that differ even slightly



Concept Review - Certificates

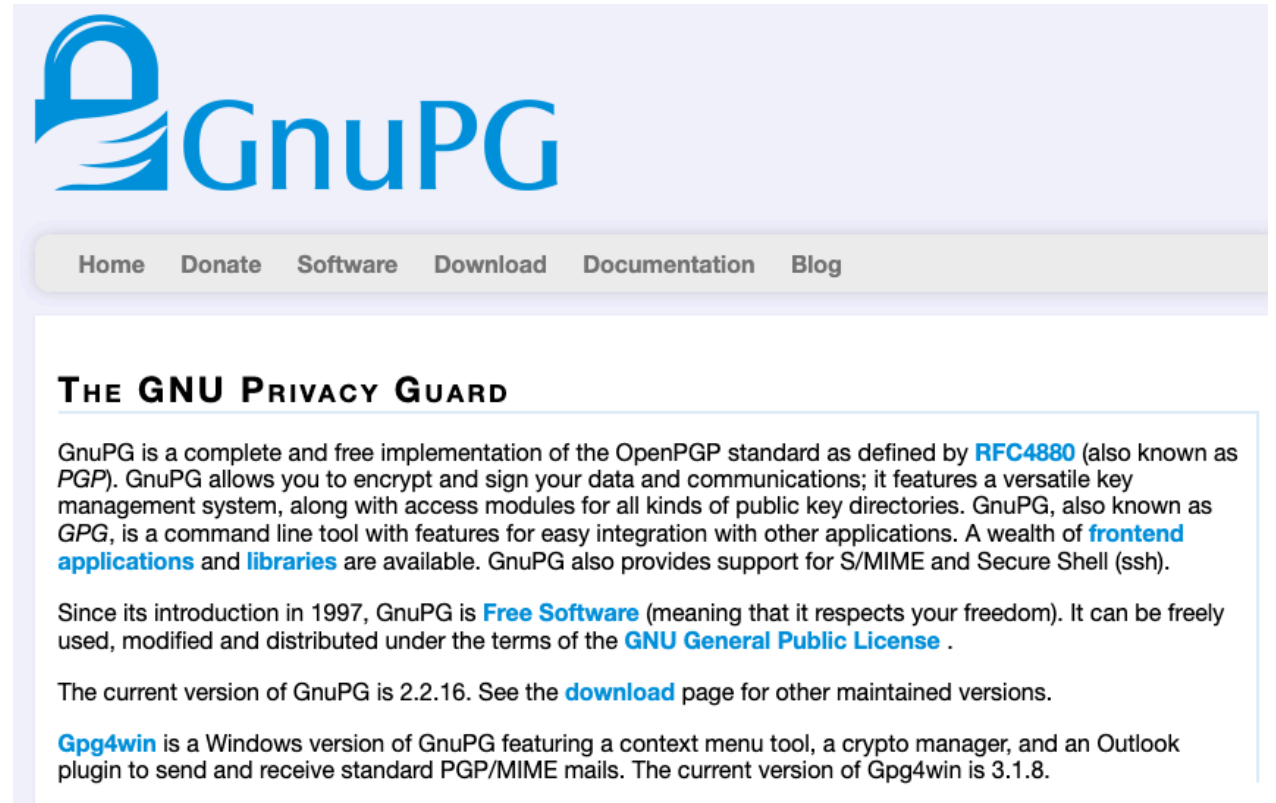
- The term *certificate* is often used interchangeably with the term *public key*.
- A certificate is a public key wrapper.
 - The public key contains the mathematical information required for crypto operations with the private key.
 - The certificate also contains attributes such as identifiers and expiration.
 - The public key and attributes are signed. The certificate contains this signature.
- Whose private key created the signature?
 - If the only signature comes from the key owner, the certificate is *self-signed*. The only way to verify is to contact owner about the fingerprint.
 - If others have signed the public key, you may already trust one of these other parties, sparing you the task of independent verification.



GPG Installation

- <https://gnupg.org>
- Select **Download**.
- Scroll to **Binary Releases**.
- Choose Platform
 - Windows
 - Simple installer is command line only (recommended for beginners).
 - Gpg4win has a GUI interface.
 - macOS
 - Better off using Homebrew.
- Verify Installation

```
$ gpg --version
gpg (GnuPG) 2.2.16
libgcrypt 1.8.4
Copyright (C) 2019 Free Software Foundation, Inc.
```



The screenshot shows the GnuPG website. At the top is the GnuPG logo, which consists of a blue stylized 'G' and the text 'GnuPG'. Below the logo is a navigation bar with links: Home, Donate, Software, Download, Documentation, and Blog. The main content area has the heading 'THE GNU PRIVACY GUARD'. Below this heading is a paragraph of text describing GnuPG as a complete and free implementation of the OpenPGP standard, defined by RFC4880. It mentions that GnuPG allows users to encrypt and sign data, manage keys, and integrate with other applications. It also notes that GnuPG provides support for S/MIME and Secure Shell (ssh). Below this paragraph is another paragraph stating that since its introduction in 1997, GnuPG is free software, meaning it respects user freedom and is distributed under the GNU General Public License. The current version is 2.2.16, and users are directed to the download page for other versions. A final paragraph mentions 'Gpg4win' as a Windows version with a context menu tool, a crypto manager, and an Outlook plugin, with the current version being 3.1.8.

GNUPG BINARY RELEASES		
OS	Where	Description
Windows	Gpg4win	Full featured Windows version of <i>GnuPG</i>
	download sig	Simple installer for the current <i>GnuPG</i>
	download sig	Simple installer for <i>GnuPG 1.4</i>
OS X	Mac GPG	Installer from the gpgtools project
	GnuPG for OS X	Installer for <i>GnuPG</i>
Debian	Debian site	GnuPG is part of Debian

pinentry.exe

- The `pinentry` program is used by GnuPG to securely obtain credentials from the user.
- On Windows, it displays a dialog box for password entry and sometimes other choices.
- It doesn't always play nice with other Windows products.

An example of overriding the pinentry mode:

```
gpg --pinentry-mode=loopback --full-generate-key
```

The password prompt should come directly from the command line rather than a pop-up window.

This shouldn't be an issue unless password entry is part of the operation.

Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing
- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

Keystore Location

- Your GnuPG installation will assume a default keystore.
 - If no keystore is found in the default location, one will be created.
 - To determine where this is, run `gpg -k`
 - This will list the default location and
 - initialize the database if it doesn't exist.
 - Observed defaults
 - **macOS:** `~/ .gnupg`
 - **Windows:** `C:\Users\<userid>\AppData\Roaming\gnupg`
- To override the default location
 - Add the option `--homedir <dir name>` to the command.
 - Add ENV variable
 - **Linux** and **macOS:** `GNUPGHOME`
 - **Windows:** Registry key: `HKCU\Software\GNU\GnuPG:HomeDir`



Tip

This is a good way to experiment through simulating multiple users. Create multiple directories and override the home directory for each command.

Generate Master Key

- Use the `--full-generate-key` option.
 - This will present several interactive options.
 - Name – full name: <first> <last>
 - Email
 - Comment – this is often left blank; could include your role; may be changed later.
 - Key strength: 4096
 - Expiration: Never
- Key material is stored in the keystore directory.

Command to get started

```
$ gpg --full-generate-key
```

Command to view results

```
$ gpg -k
```

GPG Exercise 1 – Create Master Key

1. Run: `gpg --full-generate-key`
2. Select RSA (default)
3. Select: 0 = key does not expire
4. **Real name:** Enter first and last name
5. **Email address:** Work email address
6. **Comment:** Optional; could add your department name if that's not obvious from your email address.
7. You're given a chance to modify any of the previous three entries. Once you select **Okay**, move your mouse around to generate entropy.
8. List your private key: `gpg -K`
9. List your public key: `gpg -k`

Notes

- You do **not** have to specify your entire name for the export – just enough to be unique. This applies both to portions of your email or your real name.
- The `-a` means ASCII armor. Without it, the output is binary. Placing `-a` on output commands should become habitual.

Sample Output

```
$ gpg --full-generate-key
gpg (GnuPG) 2.2.16; Copyright (C) 2019 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.

Please select what kind of key you want:
. . . (snip) . . .

$ gpg -k --fingerprint
/Users/pglezen/idsc/workshops/gpg/gpghome/pubring.kbx
-----
pub   rsa4096 2019-08-21 [SC]
      D8FA FC1B ABC7 5AA3 78E6  6579 D0F6 13FF A9C9 87D2
uid           [ultimate] Paul Glezen (Los Angeles County -
Information Systems Advisory Board) <pglezen@isab.lacounty.gov>
sub   rsa4096 2019-08-21 [E]

$
```

Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing
- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

List Public Keys –k

- List all public keys in keyring.

```
$ gpg -k
/Users/pglezen/.gnupg/pubring.kbx
-----
pub  rsa4096 2019-02-22 [C]
100687E53FFA3D550334FEE1BE8F4F17CC10208C
uid  [ultimate] Paul Glezen
<pglezen@isab.lacounty.gov>
sub  rsa4096 2019-02-22 [E]
sub  rsa4096 2019-05-21 [S]
```

Location of keyring used for the listing. In this case, it's the default location.

This is a 4096-bit RSA public key generated on 2019-02-22. It is only to be used to **[C]**ertify other keys.

This is the **fingerprint** of the public key. The `--fingerprint` option makes it more readable in groups of four hexadecimal digits.

The User ID associated with the certificate. `[ultimate]` refers to the **trust level**. I have *ultimate* trust because I generated it myself.

Subkey for **[E]**ncryption only.

Subkey for **[S]**igning only.

GPG Exercise 2 – Export Public Key

- Use `--export` option:
- Use the `-a` (ASCII Armor) parameter for text output.
- Use `--output` to specify the output file.
 - otherwise output goes to terminal; useful for
 - quick copy-n-paste into email
 - piping output to input of another command
 - common extensions are `*.gpg` and `*.pub`.
- Specify public key with unique part of
 - email
 - real name
- The bigger the key, the bigger the export.

```
$ gpg --export -a paul
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBF1dcV0BEACwMLKx/qVAtgWKUTS6Ze9/plTdE8QFdvbdf97+TKQ3aIYwVue0
BxQoiIskxXH+EHNJHCFoSLPMslf3nWYSxDfvzOCwsJfH/mwlqPbx3WxBerc6Chgd
U3lzdGVtcyBBZHZpc29yeSBCb2FyZCkgPHBnbGV6ZW5AaXNhYi5sYWNvdW50eS5n
b3Y+iQJOBBMBCAA4FiEE2Pr8G6vHWqN45mV50PYT/6nJh9IFAl1dcV0CGwMFCwkI
BwIGFQoJCAsCBBYCAwECHgECF4AACgkQ0PYT/6nJh9Iwag//TWih8p4vaTDk0RYK
wNN0HxmbWoQTkvycn9uvN19jOCcp0cSF+K2XFYWvt/cesc97DXfzOeNxZbLP7FR9
3x4XFvlsK9xdeiWxVoA6sZLx5i6YSsD4UDaU/dU+K/rPuc1lWGGotQbS8A5haRu6
sziaRMW+AdD3/4dthE11b3ZHxkFkdo9Q3E2Gdtf0AQsxG3FuSEgiKz5Hu1jPVLEDk
tamJGS6MfjmpqODCQLbby/oqZQuePbNylot4oZ2Y8DukSmu5emF7c6bgDtuhvzvc
. . . . .
09mg+DbqLQ8K012MYyuvHxIBU0LPawKl5azvpnYyU8honbzN8vg+KQrepi+OyZtb
S92LQAawJoPJ31eT5N1cVI1lzc/GxzsSusKbvD0/233ppKizt1QgfFmvktBmpg7o
BadCxOBFHywQUwj8ow==
=47A2
-----END PGP PUBLIC KEY BLOCK-----
$ gpg --export -a --output pglezen.gpg paul
$
```

GPG Exercise 3 – Import Public Key

- You need a person's public key in order to
 - encrypt an email to them
 - verify a message signed by them.
- You should try your best to verify a key before importing it.
 - the Achilles heel of GPG
 - fingerprint is most commonly used identifier.

1. Download Debbie's GPG key from
<https://lacounty-isab-gpg.s3-us-west-1.amazonaws.com/index.html>

2. Verify her public key fingerprint is

1FCB F32C 0DEA B9E0 1446 166A A44B 7759 EE7B F466

```
$ gpg --show-keys --with-fingerprint dbarton.gpg
pub  rsa4096 2019-06-01 [SC]
    1FCB F32C 0DEA B9E0 1446 166A A44B 7759 EE7B F466
uid                               Debbie Barton <dbarton@isd.lacounty.gov>
sub  rsa4096 2019-06-01 [E]
```

3. Import her key.

```
$ gpg --import dbarton.gpg
```

4. List her key to verify.

```
$ gpg -k dbarton
pub  rsa4096 2019-06-01 [SC]
    1FCBF32C0DEAB9E01446166AA44B7759EE7BF466
uid                               [ unknown] Debbie Barton <dbarton@isd.lacounty.gov>
sub  rsa4096 2019-06-01 [E]
```

Importing vs Signing

```
$ gpg -k dbarton
pub  rsa4096 2019-06-01 [SC]
    1FCBF32C0DEAB9E01446166AA44B7759EE7BF466
uid          [ unknown ] Debbie Barton <dbarton@isd.lacounty.gov>
sub  rsa4096 2019-06-01 [E]
```

- Take a closer look at the qualifier for Debbie's public key entry.
 - The [unknown] qualifier implies that we have not assigned a trust level to this key.
 - This is merely a statement about confidence in the public key itself. It's not a statement about our confidence in Debbie (that will be addressed later).
- Signing Debbie's public key means we have verified the key comes from her:
 - through a website we trust
 - through interacting with her directly over a secure medium.

GPG Exercise 4 – Sign a Key

Run: `gpg --sign-key dbarton`

This will present you with information about the key and ask you to verify with (y/N).

After signing the key, observe the difference in its entry.

```
$ gpg -k dbarton
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid: 1  signed: 1  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid: 1  signed: 0  trust: 1-, 0q, 0n, 0m, 0f, 0u
pub  rsa4096 2019-06-01 [SC]
    1FCBF32C0DEAB9E01446166AA44B7759EE7BF466
uid          [ full ] Debbie Barton <dbarton@isd.lacounty.gov>
sub  rsa4096 2019-06-01 [E]
```

The key qualifier changed from [unknown] (GPG doesn't know anything about the validity of this key) to [full], which means:

Since **you** vouch for it, GPG has complete trust in it.

Unsigned Key Usage

- You can still reference unsigned keys in your keystore.
- You receive a warning as in the example on the right.
 - Alice's key was imported.
 - An attempt is made to send an encrypted message for Alice.
 - The warning is presented; but we may still proceed.

Encrypting file with unsigned public key:

```
$ gpg -ear alice secretpasswords.txt
gpg: 1804049CFE6FD644: There is no assurance this key belongs to the named user

sub rsa4096/1804049CFE6FD644 2019-09-10 Alice Smith <asmith@xyz.lacounty.gov>
  Primary key fingerprint: 7C06 E9A1 77F0 8CCC D371 27C0 3D37 B042 1E1C 2497
  Subkey fingerprint: 00D5 8C74 C59C 06A0 8662 8877 1804 049C FE6F D644

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)
```

GPG Exercise 5 – Exchange Public Keys

Each person will be assigned a workshop attendee to whom they will email their exported public key.

1. Email your public key to the recipient. You may either send it as an attachment or copy and paste the text contents into the email body.
2. Enter the fingerprint of your public key into the WebEx chat.

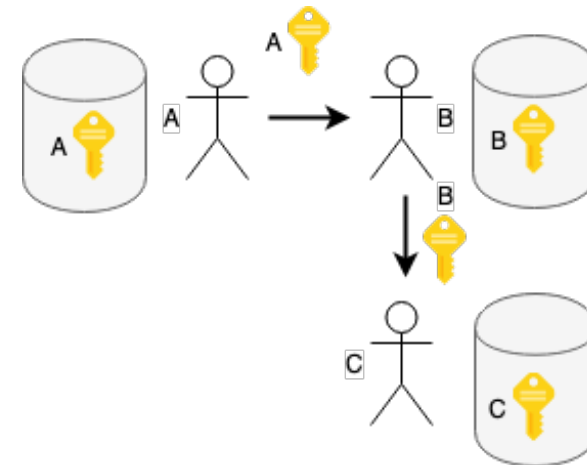
```
$ gpg --fingerprint bob
pub  rsa4096 2019-09-10 [SC]
    75E3 5118 128E F4A7 86F9  06A7 285A D454 FD8A 97B8
uid          [ultimate] Bob Jones <bjones@xyz.lacounty.gov>
sub  rsa4096 2019-09-10 [E]
```

3. Import each public key you receive.

```
$ gpg --import bob.pub
gpg: key 285AD454FD8A97B8: public key "Bob Jones
<bjones@xyz.lacounty.gov>" imported
gpg: Total number processed: 1
gpg:          imported: 1
```

4. Attempt to verify to fingerprint. Sign key if fingerprint is verified.

```
$ gpg --sign-key bob
```



Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing
- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

Encryption – Symmetric

- No personal key is used; just a pass phrase.
 - **encrypt:** `gpg -c -a <input file>`
 - `-c` is short for `--symmetric`
 - `-a` is short for `--armor` ASCII armor
 - output filename is input file + `.asc` extension.
 - when `<input file>` is absent:
 - `stdin` is used for input
 - `stdout` is used for output
 - **decrypt:** `gpg -d <encrypted file>`
 - `-d` is short for `--decrypt`
- Encrypted message is easily included in emails or messaging applications.

```
$ cat secretpasswords.txt
userid: importantuser
passwd: logmein
$ gpg -c -a secretpasswords.txt
$ cat secretpasswords.txt.asc
-----BEGIN PGP MESSAGE-----

jA0EBwMCVDP4FaN9Q3zq0mgBVdLRIGMQ10YrBtJ5V+1uSfLImTw1wCgkD0MK00gd
KoiT1hA3I5Rm46cCqNrOWK3opvoN+3Jw4PzUNYmNeZ9lM6LlkIzr1dGUtuB5q1B9
zhSj+aUssGFdTL/5JU9FJWMD43muFAOwg==
=fdfr
-----END PGP MESSAGE-----
$ gpg -d secretpasswords.txt.asc
gpg: AES encrypted data
gpg: encrypted with 1 passphrase
userid: importantuser
passwd: logmein
```

Notes:

- Creating an encrypted file does **not** delete the plain text file.
- GPG will cache your session key for a while.

Encryption - Asymmetric

- Requires the public key of the recipient; no pass phrase is involved.
 - **encrypt:** `gpg -e -a -r <recipient>`
 - `-e` is short for `--encrypt` (asymmetric encryption)
 - `-a` is short for `--armor`
 - `-r` is short for `--recipient`
 - can all be combined as `-ear`
 - **decrypt:** `gpg -d <encrypted file>`
 - `-d` is short for `--decrypt`
 - You can only decrypt a message for which you possess the private key.
 - Since the key id is stored in the message, this command will automatically use the correct private key from your keystore if you have it.

```
$ gpg -k
/Users/pglezen/idsc/workshops/gpg/gpghome/pubring.kbx
-----
pub   rsa4096 2019-08-21 [SC]
      D8FAFC1BABC75AA378E66579D0F613FFA9C987D2
uid           [ultimate] Paul Glezen (Los Angeles County - Information Systems
Advisory Board) <pglezen@isab.lacounty.gov>
sub   rsa4096 2019-08-21 [E]

pub   rsa4096 2019-06-01 [SC]
      1FCBF32C0DEAB9E01446166AA44B7759EE7BF466
uid           [ full ] Debbie Barton <dbarton@isd.lacounty.gov>
sub   rsa4096 2019-06-01 [E]

$ gpg -ear Deb secretpasswords.txt
$ ls -l secretpass*
 38 Aug 27 15:13 secretpasswords.txt
927 Aug 30 14:13 secretpasswords.txt.asc
228 Aug 27 15:14 secretpasswords1.txt.asc
```

Notes

- The recipient string only has to be large enough to distinguish it from other recipients (whether email, name, or key id).
- The new asymmetrically encrypted file is three times larger than the symmetrically encrypted file from three days earlier (927 bytes vs 228 bytes). That's because the asymmetric result includes information about the key used to encrypt the message. For larger input messages, this difference is not as stark.

Decryption – Asymmetric

- The example on the right shows Debbie decrypting the attachment with her private key.
 - The email attachment was downloaded.
 - She ran the command in red.
 - GPG prints the key used to encrypt the message.
 - The contents (the last two lines) are sent to standard output.
 - This could be piped to another program or a files.
 - Use the `--output` flag to specify an output filename.

```
C:\Users> gpg --decrypt secretpasswords.txt.asc
gpg: encrypted with 4096-bit RSA key, ID DEBC09015FED7075,
created 2019-06-01
        "Debbie Barton <dbarton@isd.lacounty.gov>"
userid: importantuser
passwd: logmein
```

GPG Exercise 6 – Asymmetric Encryption

Each person will encrypt a message to to the workshop participant that sent them a public key.

1. Create a sample text file named `sample.txt` with your name on the first line and your email address on the second line.

2. Encrypt the sample using the public key you received in Exercise 5.

```
$ gpg --output jack.sample.asc -ear bob sample.txt
```

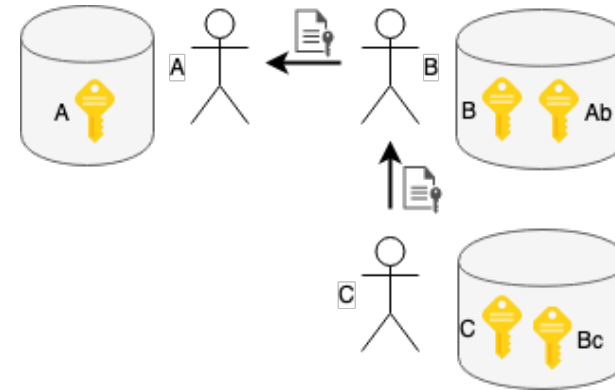
Note: The sample message will have different encrypted outputs for different public keys.

3. Email the encrypted file. You may either

- attach the file, or
- paste the text contents of the ASCII armor file into the email body.

4. Decrypt the messages you receive from others using your public key.

```
$ gpg -d jack.sample.asc
```

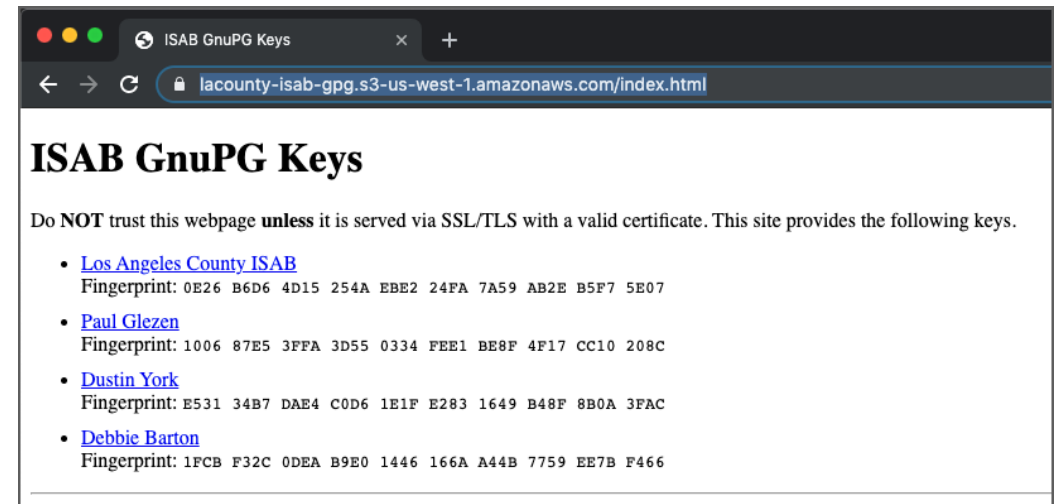


Publish Key Signers

- A network of public key servers exist to which one can
 - publish their public key and
 - download public keys of others.
- OpenPGP key servers:
 - Do not allow you to delete keys that you previously published.
 - You can **update** a key server with **merged** information (e.g. new signers or certificate details).
 - ISAB is investigating the potential for hosting an LA County internal key server.

- While adoption of GPG is still light:
 - Keys and their fingerprints are published to a webpage hosted by ISAB.

<https://lacounty-isab-gpg.s3-us-west-1.amazonaws.com/index.html>

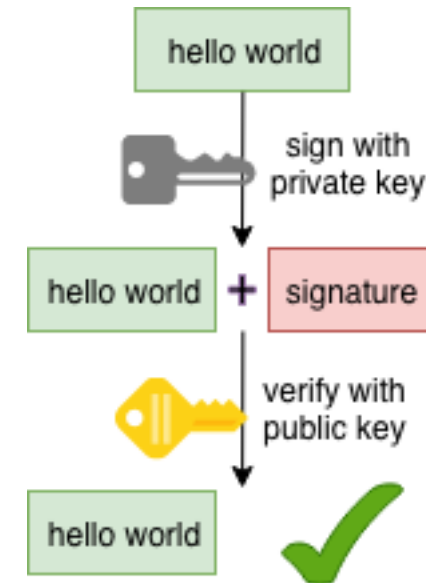


Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing
- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

Signatures

- There are two types of signatures in GPG that are sometimes confused.
 - **Key** signing: vouching for another person's public key; `--sign-key`, `--lsign-key`
 - **File** signing: vouching for the contents of a file; `--detach-sign`, `-b`
- We've already addressed the first type. The next few slides will address the second type.
 - The goal is to vouch for the contents of a file.
 - Any attempted to compromise the file will be detectable.



GPG Exercise 7 – Detached Signature

- Detached means that the signature is separate from the rest of the message.
 - **Pro** – Easier to disregard for people that don't need to verify.
 - **Con** – Requires that validator knows exactly what to verify.
- Consider the example on the right:
 - publishing a list of URLs for downloads
 - URLs are **not** secret
 - But if they were changed, people would download from the **wrong** site.

Alice signs the list of legitimate URLs with her private key.

```
$ cat downloads.txt
These are the official download sites:
https://aaal.isab.lacounty.gov/download/data1.csv
https://bbbl.isab.lacounty.gov/download/data1.csv
$ gpg -b -a --output downloads.sig downloads.txt
$ cat downloads.sig
-----BEGIN PGP SIGNATURE-----

iQIzBAABCAAdFiEEfAbpoXfwjMzTcSfAPTewQh4cJJcFA114B50ACgkQPTewQh4c
JJef6hAAAtEXUK87yLmuJ1uLwjCZzhgU14dzVRzHzgKX2dAAD1vIRVzf9aFEERkdJ
. . . . . snip . . . . .
+ytRVmlu3qhcWfGhnLD7Ohhycru+iejj1dkHF1tzrmYZPZCs0e4QXXbSKLBjQDJr
0yrPKB2QdGGhAJDUXwmIGbTjkyYwlRnceWvQpT8bI/PxNFWMISM=
=17Fr
-----END PGP SIGNATURE-----
```

Attempt to verify.

```
$ gpg --verify downloads.sig downloads.txt
gpg: Signature made Tue Sep 10 13:29:17 2019 PDT
gpg: using RSA key 7C06E9A177F08CCCD37127C03D37B0421E1C2497
gpg: Good signature from "Alice Smith <asmith@xyz.lacounty.gov>" [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: 7C06 E9A1 77F0 8CCC D371 27C0 3D37 B042 1E1C 2497
```

We get a notification that we still haven't signed Alice's public key yet. But the signature itself is valid.

Real Digital Signatures

In this webpage screenshot, Amazon Web Services (AWS) is providing links to their CloudWatch package. In the right column is the GPG signature for each download.

Platform	Download Link	Signature File Link
Amazon Linux and Amazon Linux 2	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.amazonaws.com/amazoncloudwatch-agent/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
	https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Putty used to be the most popular way to use SSH on Windows (before OpenSSH). Note the **signature** link next to each download link.

putty.exe (the SSH and Telnet client itself)			
32-bit:	putty.exe	(or by FTP)	(signature)
64-bit:	putty.exe	(or by FTP)	(signature)
pscp.exe (an SCP client, i.e. command-line secure file copy)			
32-bit:	pscp.exe	(or by FTP)	(signature)
64-bit:	pscp.exe	(or by FTP)	(signature)

The download page for the popular Nginx web/reverse proxy server posts a signature link next to each download link.

nginx: download			
Mainline version			
CHANGES	nginx-1.17.3	pgp	nginx/Windows-1.17.3
Stable version			
CHANGES-1.16	nginx-1.16.1	pgp	nginx/Windows-1.16.1
Legacy versions			
CHANGES-1.14	nginx-1.14.2	pgp	nginx/Windows-1.14.2
CHANGES-1.12	nginx-1.12.2	pgp	nginx/Windows-1.12.2
CHANGES-1.10	nginx-1.10.3	pgp	nginx/Windows-1.10.3

Topics

- Crypto Review
- GPG Installation
- Master Key Generation
- Key Management
 - Key Viewing
 - Key Import/Export
 - Key Signing
- Encryption
- Decryption
- Message Signatures
- Validation
- Trust

Key Signing

- In previous slides we signed files to vouch for their content.
- We also signed keys to vouch for their authenticity (last step of exercise 5).

Review

Bob has imported Alice's certificate – but has not yet indicated that he trusts it.

```
$ gpg -k alice
pub  rsa4096 2019-09-10 [SC]
    7C06E9A177F08CCCD37127C03D37B0421E1C2497
uid          [ unknown ] Alice Smith <asmith@xyz.lacounty.gov>
sub  rsa4096 2019-09-10 [E]
```

After calling Alice on the phone and verifying the finger print, Bob feels confident enough to sign the key he imported.

```
$ gpg --sign-key alice
pub  rsa4096/3D37B0421E1C2497
    created: 2019-09-10  expires: never      usage: SC
    trust: unknown      validity: unknown
Primary key fingerprint: 7C06 E9A1 77F0 8CCC D371 27C0 3D37
B042 1E1C 2497
    Alice Smith <asmith@xyz.lacounty.gov>
Are you sure that you want to sign this key with your
key "Bob Jones <bjones@xyz.lacounty.gov>" (285AD454FD8A97B8)

Really sign? (y/N) y
```

Bob is prompted for his private key's password before it can signed.
After Alice's key is signed, it's displayed as follows.

```
$ gpg -k alice
pub  rsa4096 2019-09-10 [SC]
    7C06E9A177F08CCCD37127C03D37B0421E1C2497
uid          [ full ] Alice Smith <asmith@xyz.lacounty.gov>
sub  rsa4096 2019-09-10 [E]
```

Verifying Alice's signature on downloads.txt does not yield a warning.

```
$ gpg --verify downloads.sig downloads.txt
gpg: Signature made Tue Sep 10 13:29:17 2019 PDT
gpg:      using RSA key
    7C06E9A177F08CCCD37127C03D37B0421E1C2497
gpg: Good signature from "Alice Smith <asmith@xyz.lacounty.gov>" [ full]
```

Bob exports Alice's key for Chris to import.

```
$ gpg --export --output alice-from-bob.pub alice
```

Of what benefit is it for Bob to export Alice's key for Chris? Why not just send Chris the *original* public key file Bob received from Alice?

```
$ dir *.pub
-rw-r--r--  1 pglezen  staff   3902 Sep 10 15:10 alice-from-bob.pub
-rw-r--r--  1 pglezen  staff   3139 Sep 10 08:53 alice.pub
```

Note the difference in file sizes. Bob's export contains something that Alice's original export lacks; it contains Bob's signing of Alice's key. If Chris receives `alice-from-bob.pub`, he has two options for verification:

- verify the fingerprint with Alice (just like Bob did), or
- verify Bob's signature (which Chris might already *trust*).

Trust

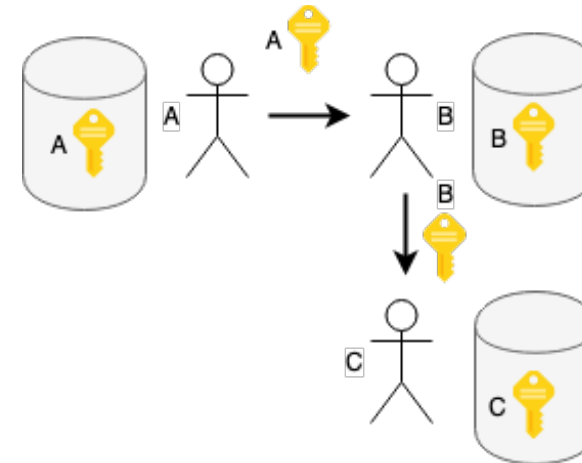
- The last word of the last slide is *trust*, not "verify," which is the term we typically use for signatures.
 - Of course, the signature must be verified to proceed with the consideration. But the question of **trust** pertains to the **human** to whom the signing key belongs.
 - Is this person trustworthy?
 - Is this person honest, but cavalier about signing the keys of others?
 - Does this person understand what it means to sign another person's key?
 - In GPG terminology **trust** is a human factor, not a cryptographic concept.
- GPG defines these levels of trust:
 - **Ultimate** – This means yourself. For example, you have ultimate trust in your own public key.
 - **Full** – Complete trust in another person.
 - **Unknown** – This is *the default* for newly imported keys. It means you have not yet assigned a trust level to the key (owner).
 - **Never** – Identical in level to Unknown, but is explicitly set by the keystore owner.
 - **Marginal** – Means that the key owner is marginally trustworthy.
 - Unfortunately, the human trust level is not shown in output from `-k` (`--list-keys` command). What we see in square brackets is the *validity* setting (which is our trust in the key itself, **not** in the human represented by the key).

GPG Exercise 8 – Trust: Step 1

Note: Step 1 may be skipped if you already did Exercise 5.

Each participant in this lab should be assigned a recipient. The recipient is the person that will receive keys from you. In the diagram on the right, B is the recipient for A; C is the recipient for B.

1. Each participant starts with their own public key in their public key store. Add a message to the WebEx chat with your key's fingerprint.
2. Each participant exports their public key to a file with a .gpg extension. Be sure to use the `-a` option for ASCII armor.
3. Each participant emails his/her public key to assigned recipient. (There is no need to respond to the sender).
4. Each recipient downloads the received public key to their working directory.



GPG Commands

```
gpg --fingerprint <key name>
gpg --export -a --output <file name> <key name>
```

GPG Exercise 8 – Trust: Step 2

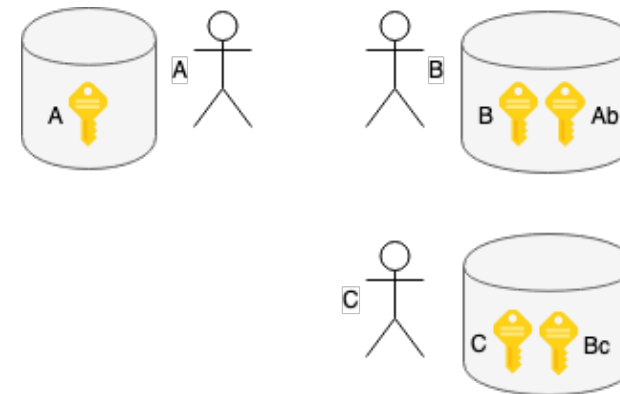
Note: Step 2 may be skipped if you already did Exercise 5.

Each participant imports key from sender, verifies fingerprint, then signs key.

1. Import the key into your key store.
2. Use the chat section of the WebEx to verify the fingerprint.
3. Sign the key with the verified fingerprint. This will require you to provide the password to your private key.

GPG Commands

```
gpg --import <key file name>  
gpg --fingerprint <key name>  
gpg --sign-key <key name>
```



Ab = Alice's key signed by Bob

Bc = Bob's key signed by Chris

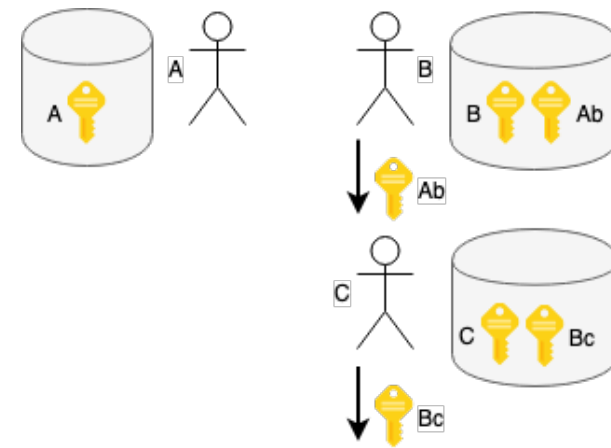
GPG Exercise 8 – Trust: Step 3

Each participant exports the key they received and signed; then sends it to their designated recipient.

1. Export the key you signed in the previous step. For the file name, use the convention `alice-from-bob.gpg`.
2. Email exported key to recipient. Bob (in this context) would email Alice's key to Chris.

GPG Command

```
gpg --export --output <key file name> <key name>
```



Ab = Alice's key signed by Bob

Bc = Bob's key signed by Chris

GPG Exercise 8 – Trust: Step 4

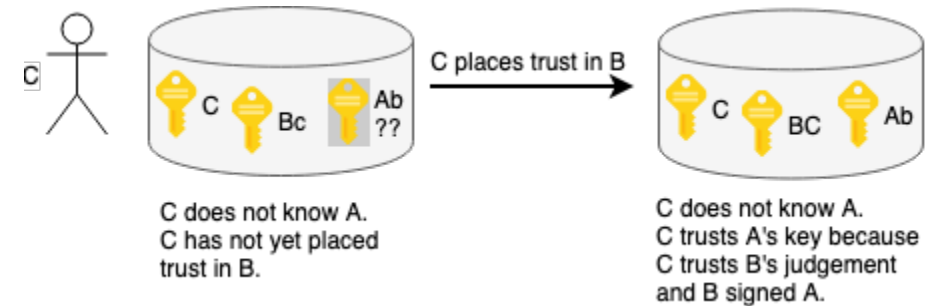
Assume Chris (C) has no way to verify Alice's key directly from Alice. It's signed by Bob and Chris trusts Bob's judgement. Chris needs to indicate to GPG that Chris trusts Bob.

1. Chris imports `alice-from-bob.gpg`. At this point, the validity of Alice's key should show `[undef]`.
2. Chris edit's Bob's key to indicate trust using `--key-edit`.
3. After Chris trusts Bob, Alice's key validity should be `[full]`.

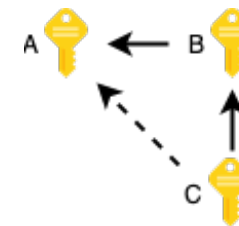
GPG Commands

```
$ gpg --import alice-from-bob.gpg
$ gpg --fingerprint
$ gpg --key-edit bob
gpg> trust
Your decision? 4
gpg> quit
$ gpg --fingerprint
```

Interactive GPG `--key-edit` session that uses the `trust` and `quit` subcommands.



Bc = Bob's key signed but not trusted by Chris
BC = Bob's key signed and trusted by Chris.
Ab = Alice's key signed by Bob.



Solid arrows represent manual key signing. Dashed arrow occurred through C trusting B.

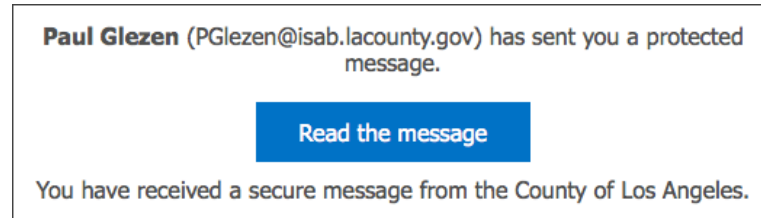
Review

- We've just covered some GPG basics. And yet we can
 - Create our own public/private key pair
 - Export our public key for others.
 - Import the public key of others
 - Sign their key to signify verification.
 - Encrypt with
 - symmetric encryption (shared secret)
 - asymmetric encryption (recipient's public key)
 - Digitally sign content.
 - Verify signature of others.
- There is much more to investigate.
 - sub-keys allow you to separately generate keys for encryption and signatures; customize their expiration dates.
 - key servers make key exchanges much easier than the manual email exchanges in our exercises.
 - trust models key be extended beyond absolute I do/I don't trust.
 - One can be "marginally trusted".
 - Several algorithms are supplied to determine how multiple "marginal signatures" constitutes a full trust.
 - Many email clients provide GnuPG support. Other products exist to provide GnuPG support as a plugin.
 - Sign Git commits and tags with your GPG key.

Microsoft Outlook Support

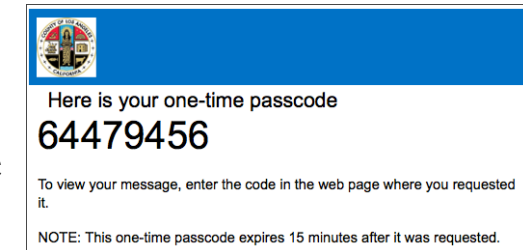
Microsoft Outlook provides some encryption support. The following screenshots show how this looks from a **non-Outlook** client.

1. I receive the following message at my home email address.

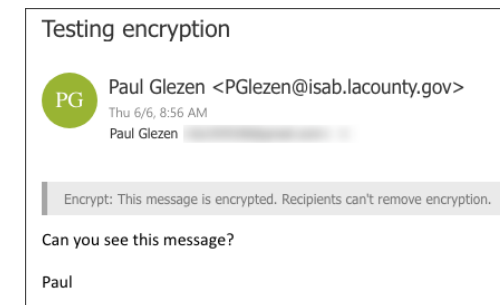


2. Clicking the blue button takes me to a web page that offers me the choice of
 - Signing in through Google
 - Authenticate with one-time passwordI choose the second option.

3. I choose the second option. I receive a second email with the contents below.



4. Entering the code in the web page reveals the email contents.



GPG and Git

- You can sign Git commits and tags with your GPG key.

- Tell Git which key to use.

```
$ gpg --list-secret-keys --keyid-format LONG
/Users/pglezen/.gnupg/pubring.kbx
-----
sec   rsa4096/BE8F4F17CC10208C 2019-02-22 [C]
      100687E53FFA3D550334FEE1BE8F4F17CC10208C
uid           [ultimate] Paul Glezen
      <pglezen@isab.lacounty.gov>
ssb   rsa4096/96CBDBD6BEE0B116 2019-02-22 [E]
ssb   rsa4096/93C50A1DEC07638A 2019-05-21 [S]

$ git config --global user.signingkey BE8F4F17CC10208C
```

- The following setting may be necessary to enter your key password

```
$ export GPG_TTY=$(tty)
```

This command will create a tag and sign it.

```
$ git tag -s v0.1.7
```

(Not shown: password prompt for private key and tag comment)

The tag can later be verified.

```
$ git verify-tag v0.1.7
gpg: Signature made Mon Aug 26 12:02:44 2019 PDT
gpg:                using RSA key A4E3ECD7E60E2878931D7D0C93C50A1DEC07638A
gpg: Good signature from "Paul Glezen <pglezen@isab.lacounty.gov>" [ultimate]
```

Notes:

- The [ultimate] label is shown above because I'm verifying a signature created from my own key.
- By default, commits are **not** signed. Commit signatures can be enabled with `git config commit.gpgSign true`. This can become onerous after a while; usually not done.
- A single commit can be signed with the `-S` flag to the commit command.

Questions

