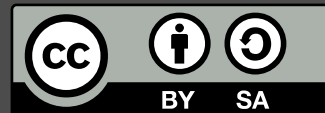


GPG, SSH, and Identity for Beginners

GUADEC Strasbourg, July 2014

**Federico Mena Quintero
federico@gnome.org**

**263F 590F 7E0F E1CB 3EA2
74B0 1676 37EB 6FB8 DCCE**





DEMO



- **Crypto is **Hard(tm)****
- **Be diligent...**
- **... even if you aren't an expert**
- **It's a prophylactic**

- This is not a talk about **OPSEC**
- **National Operations Security**
- **Spy stuff - “tradecraft”**
- **Useful to covert organizations in general**



Public-key Cryptography



- **Symmetric-key cryptography**
- **Easy to understand**
- **Key distribution is a problem**

- **Private key**

- The physical key
- The seal
- **DO NOT SHARE**



- **Public key**

- The lock/box
- The imprint
- **SHARE**



Confidentiality

Data integrity

Authentication

Non-repudiation

What can we do?

Your private key



Their public key



Email

Send an email

Encrypt
with their public key
("This is for **you** only")
Lock it in a box



Sign
with your private key
("This comes from **me**")
Stamp it with my seal

Receive an email



Decrypt
with your private key
("Only *I* can read this")
Unlock the box

Authenticate
with their public key
("I'm assured *you*
wrote ***exactly this***")
Check the imprint



Distributing packages

Distribute a package



Sign
with your private key
 (“*This* comes from *me*”)
Stamp it with my seal

Distribute a package

Let them ensure
this comes from you

Let them validate
the tarball's integrity

Your announcement email:

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

```
Hash: SHA1
```

```
Hi everyone!
```

```
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcd6f9efceab68c746.
```

```
Sincerely,
```

```
J. Random Hacker
```

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

```
Version: GnuPG v1
```

```
iQGVAwUBU45IaJEDl9iNKTGaAQKcAv8CFVTBvbN0u
```

```
...
```

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



Sign

with your private key
("This comes from me")

Stamp it with my seal

Receive a package

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

```
Hash: SHA1
```

```
Hi everyone!
```

```
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbd6f9efceab68c746.
```

```
Sincerely,  
  J. Random Hacker
```

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

```
Version: GnuPG v1
```

```
iQGVAwUBU45IaJEDl9iNKTGaAQKCvAv8CFVTBvbN0u
```

```
...
```

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

Receive a package

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1  
  
Hi everyone!  
  
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbd6f9efceab68c746.  
  
Sincerely,  
  J. Random Hacker  
  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1  
  
iQGVAwUBU45IaJEDl9iNKTGaAQKCvAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

Authenticate
with their public key
(“I’m assured *you*
packaged *exactly this*”)
Check the imprint



Receive a package

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1  
  
Hi everyone!  
  
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbd6f9efceab68c746.  
  
Sincerely,  
  J. Random Hacker  
  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1  
  
iQGVAwUBU45IaJEDl9iNKTGaAQKCvAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

Your email client checks the signature

You check the tarball's hash

Authenticate
with their public key
(“I'm assured *you*
packaged *exactly this*”)
Check the imprint



Generating key-pairs

Your private key



Your public key



Generating key-pairs

Your private key

Your public key



Distributing **public** keys

How do you
give these
to people?



Receiving **public** keys

How do you
find/receive
these?



How do you
ensure the
keys are
authentic?



Trust

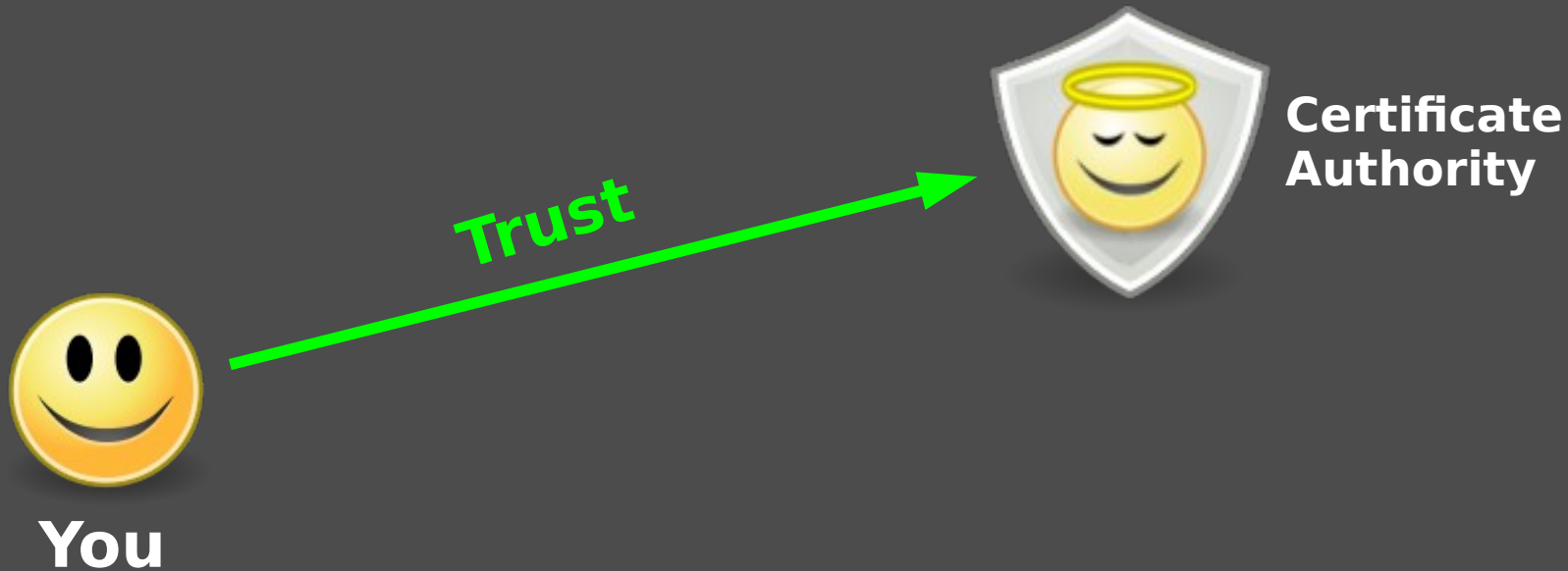
Trust models

Centralized model (Certificate Authorities)

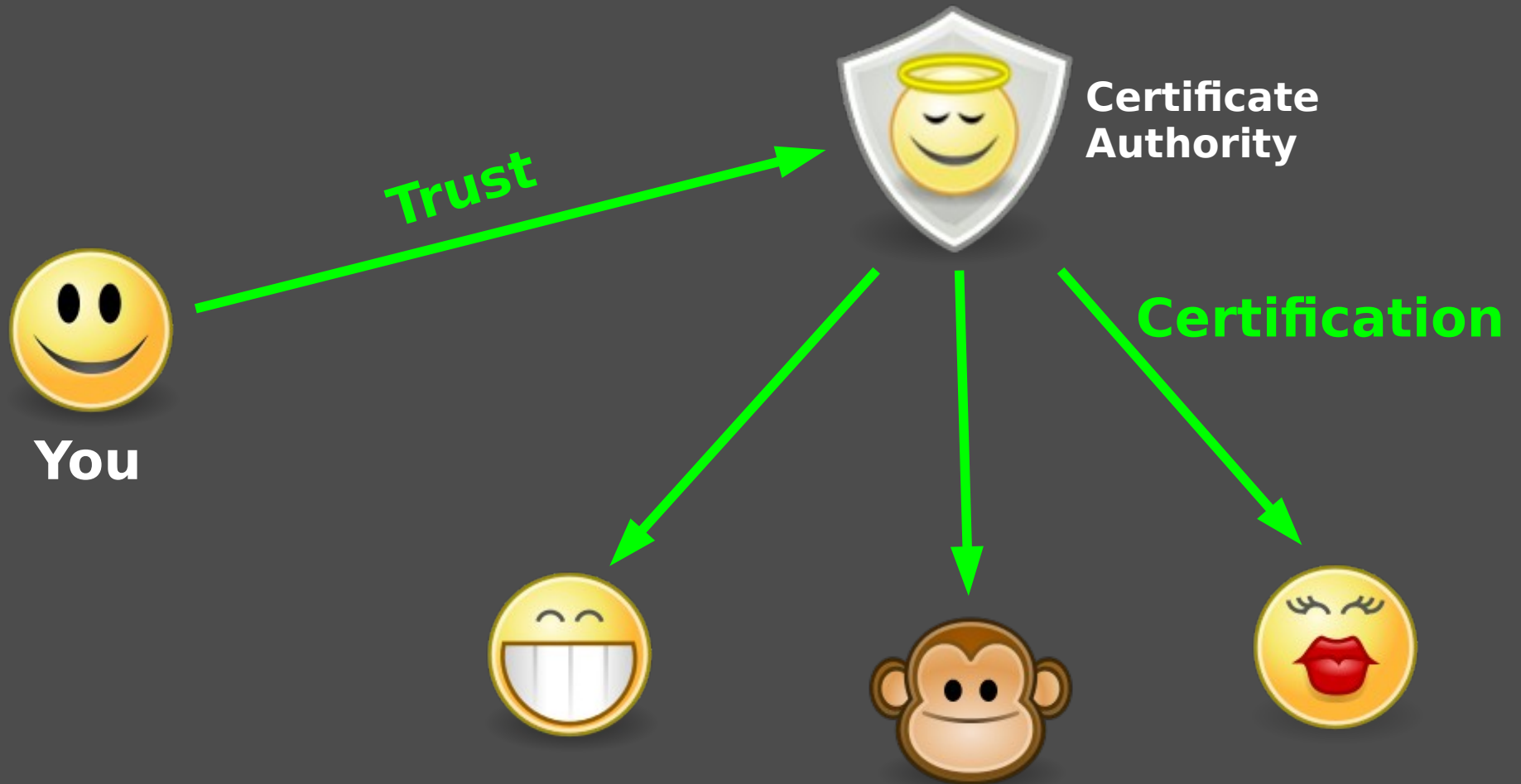


You

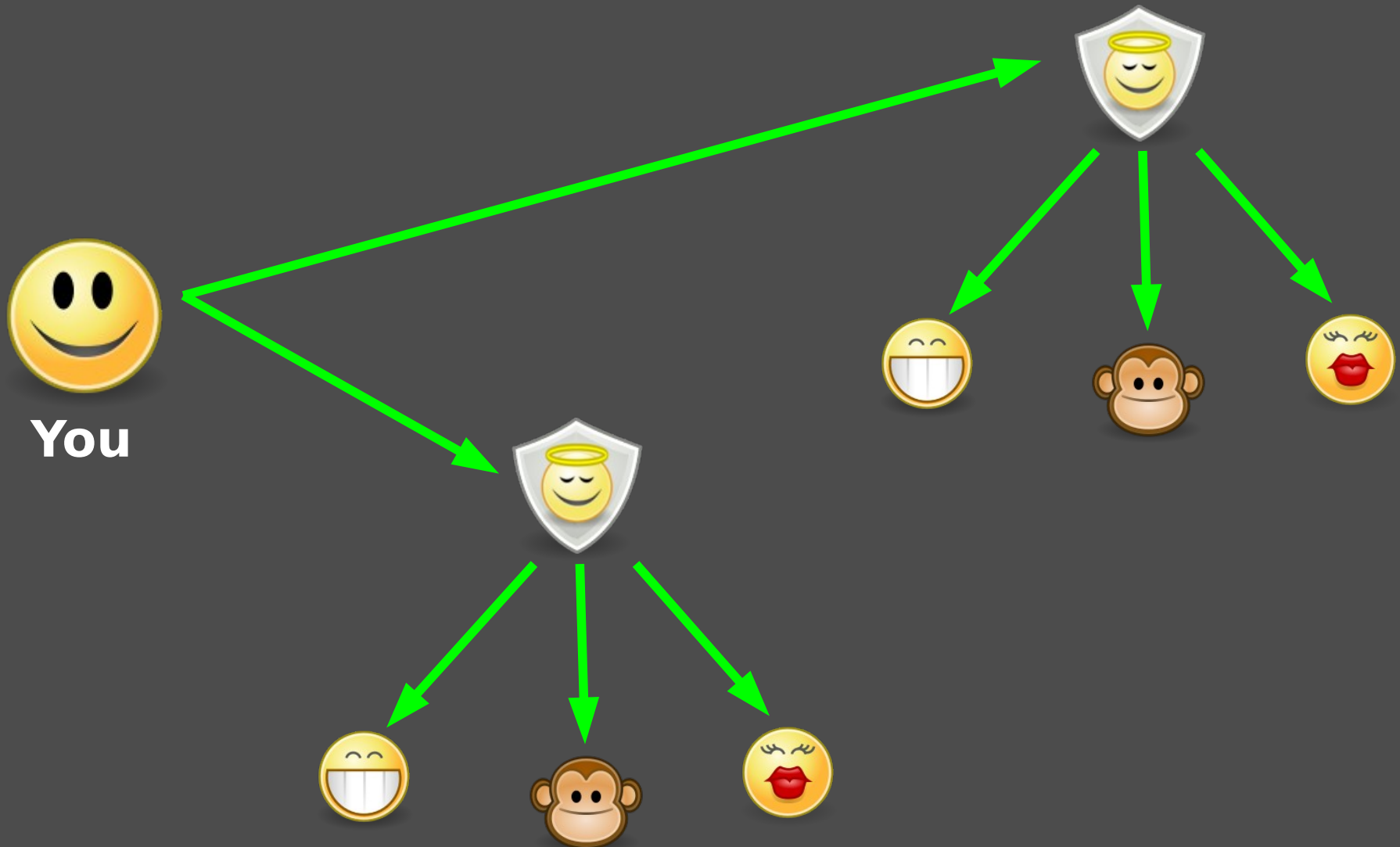
Centralized model (Certificate Authorities)



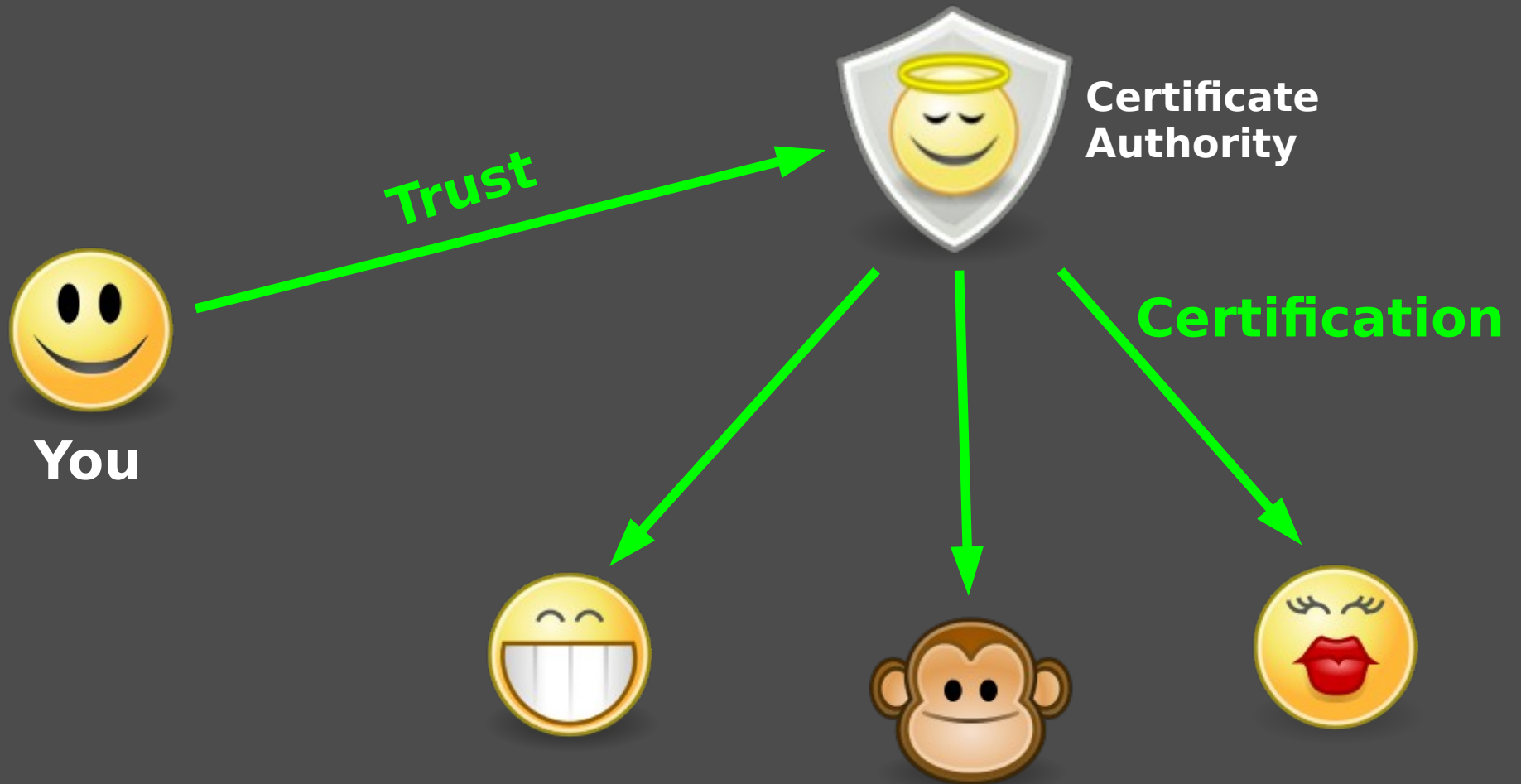
Centralized model (Certificate Authorities)



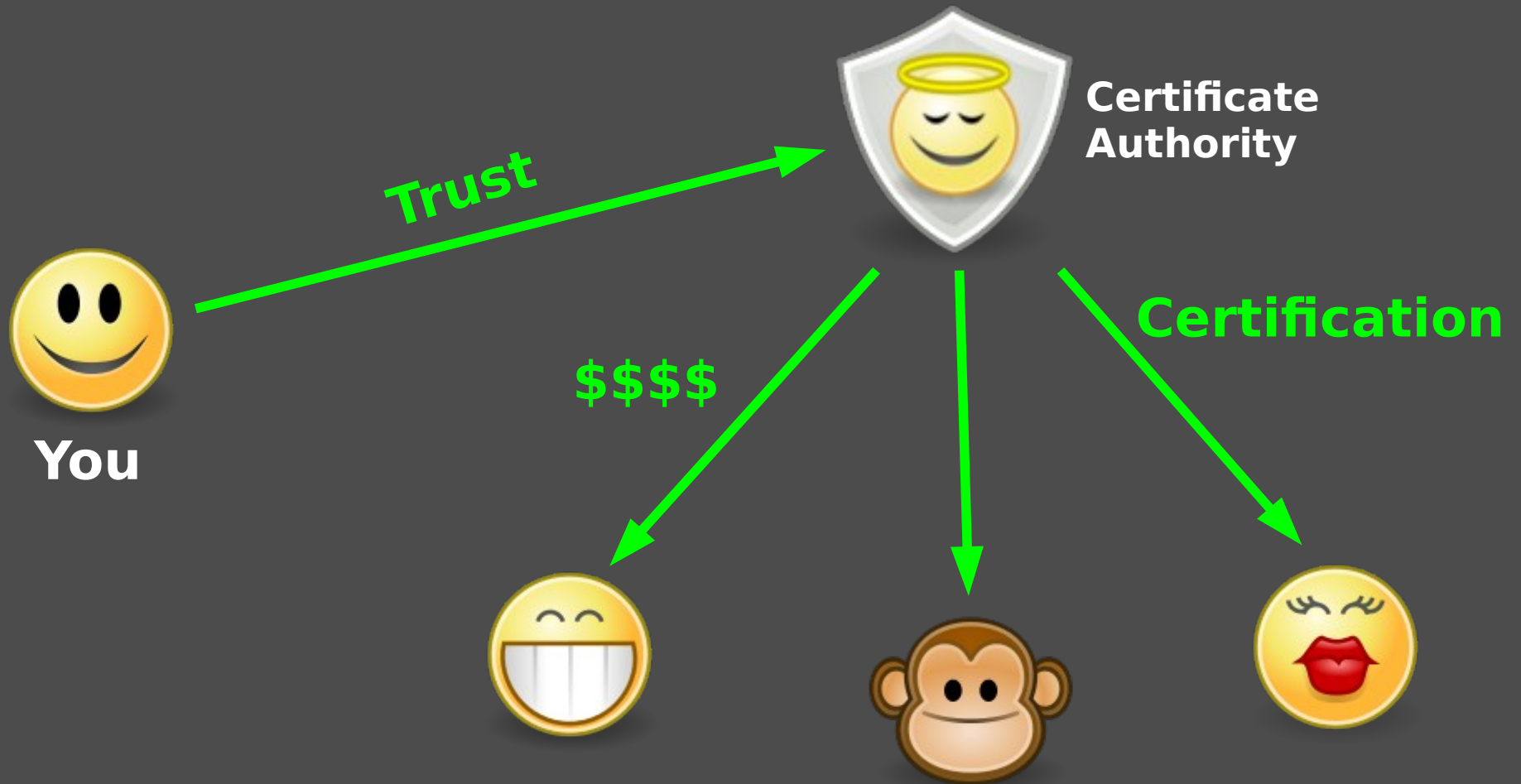
Centralized model (Certificate Authorities)



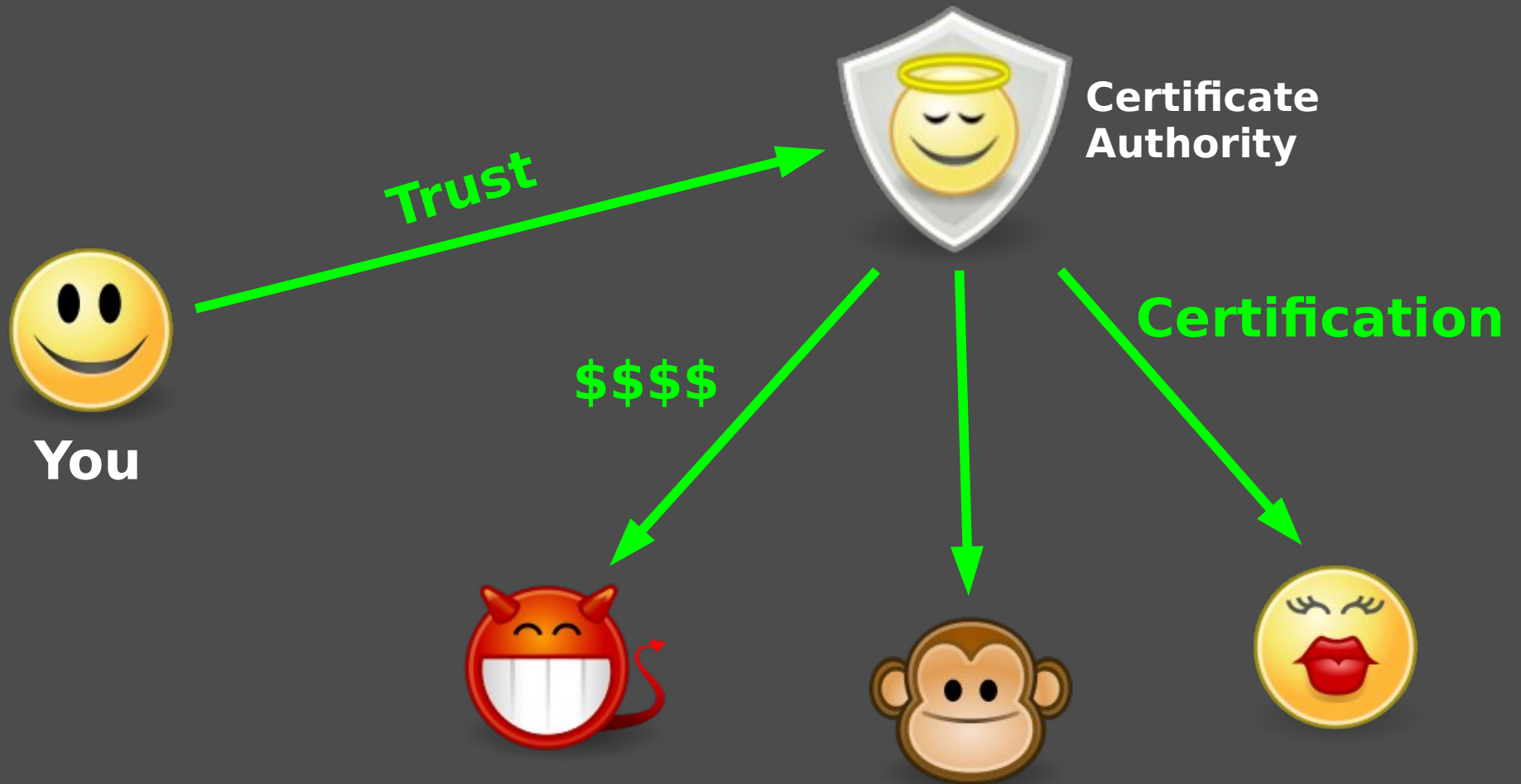
Centralized model (Certificate Authorities)



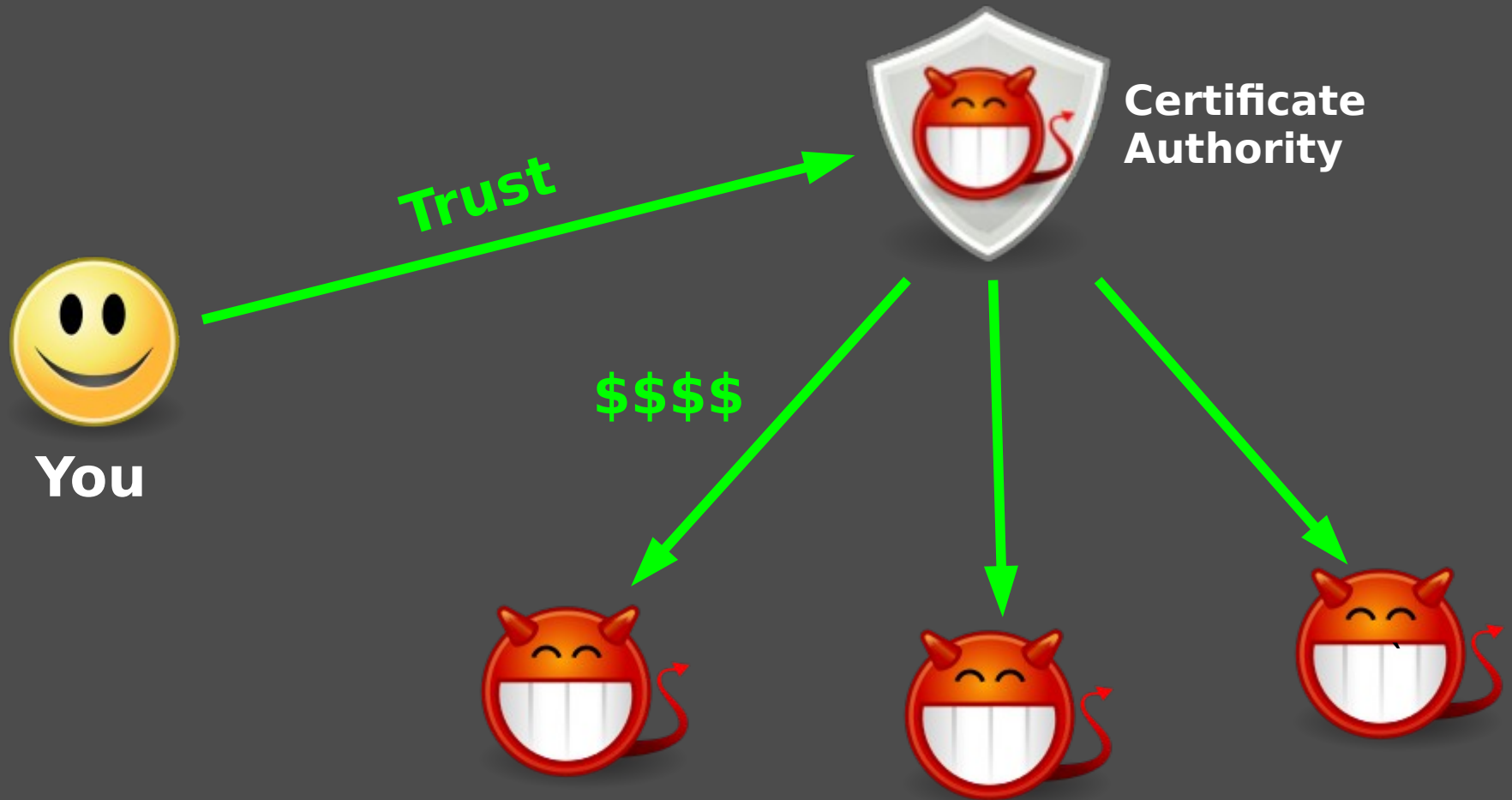
Centralized model (Certificate Authorities)



Centralized model (Certificate Authorities)



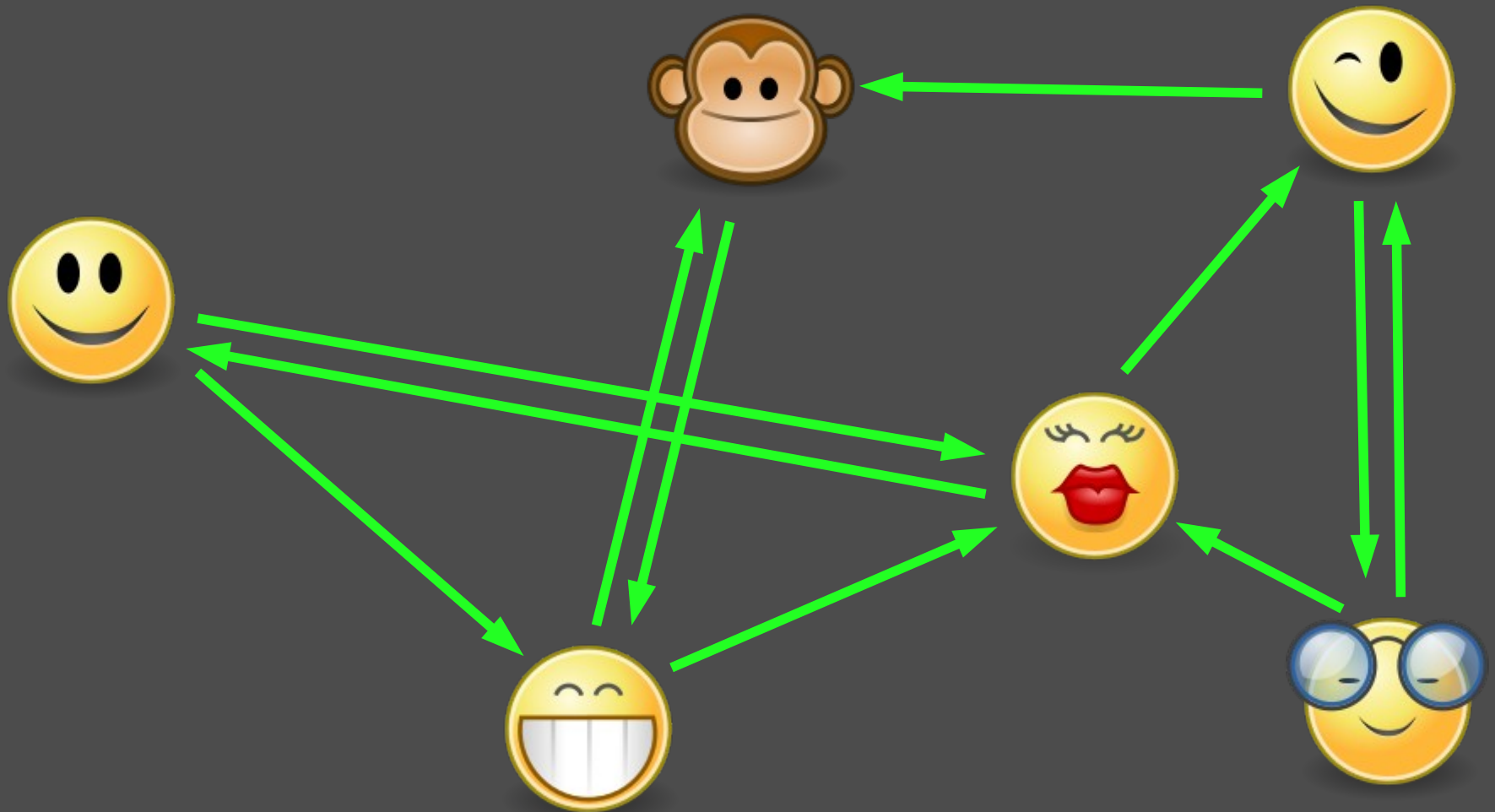
Centralized model (Certificate Authorities)



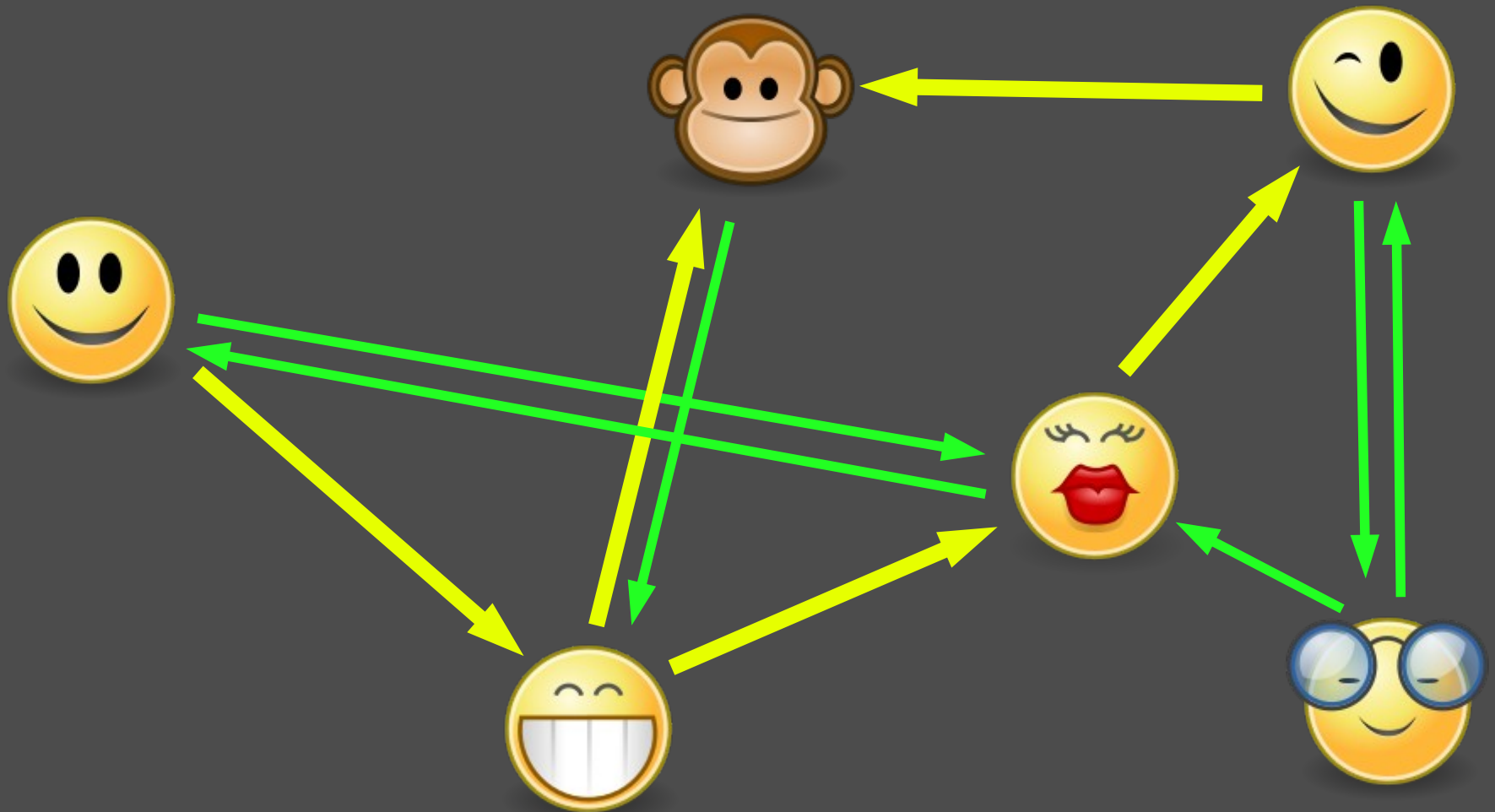


You

Distributed model (Web of Trust)



Do you trust the monkey?



SSH
(secure shell)

```
$ telnet somehost.example.com  
Trying 192.168.1.87...
```

```
somehost login: _
```



```
$ telnet somehost.example.com  
Trying 192.168.1.87...
```

```
somehost login: federico  
Password: _
```

```
$ telnet somehost.example.com  
Trying 192.168.1.87...
```

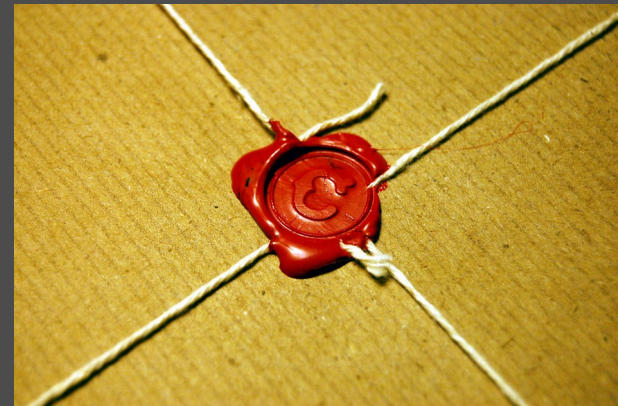
```
somehost login: federico  
Password: _
```

KABOOM

SSH user (you)

- **Private key**
(yours only)
 - Encrypted with a *passphrase*

- **Public key**
 - Copied to
~/.ssh/authorized_keys
in destination
machines

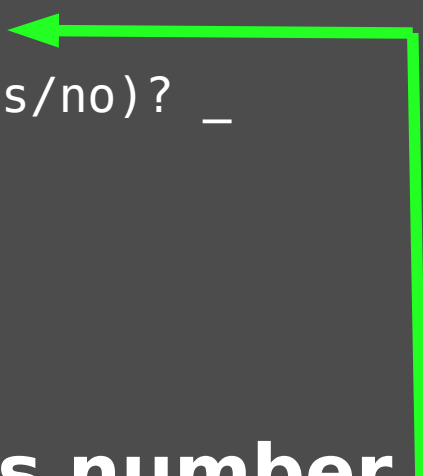


SSH server (git.gnome.org)

- **Private key**
(server's only)
- **Public key**
 - Copied to `~/.ssh/known_hosts` in your machine, automatically, when you first ssh in

Am I connecting to the correct machine?

```
pambazo:~$ ssh tlacoyo.local
The authenticity of host 'tlacoyo.local (192.168.1.87)' can't be
established.
ECDSA key fingerprint is
fc:26:07:61:c9:2c:1f:6c:90:64:59:d7:11:6d:6f:06.
Are you sure you want to continue connecting (yes/no)? _
```



Ask your sysadmin about this number

Hint:

```
tlacoyo:~$ ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key
256 fc:26:07:61:c9:2c:1f:6c:90:64:59:d7:11:6d:6f:06 root@linux-szu1 (ECDSA)
```



Enter password to unlock the private key

An application wants access to the private key
'federico@guanabana.ximian.com', but it is locked

Password:

Automatically unlock this key whenever I'm
logged in

Cancel

Unlock



Enter password to unlock the private key

An application wants access to the private key
'federico@guanabana.ximian.com', but it is locked

Password:

Automatically unlock this key whenever I'm
logged in

Cancel

Unlock

WHY?

```
pambazo:~/src$ git clone ssh://git.gnome.org/git/seahorse
Cloning into 'seahorse'...
The authenticity of host 'git.gnome.org (209.132.180.184)' can't be
established.
RSA key fingerprint is
00:39:fd:1a:a4:2c:6b:28:b8:2e:95:31:c2:90:72:03.
Are you sure you want to continue connecting (yes/no)? _
```

Do we publish this anywhere?

GPG
(GNU Privacy
Guard)

PGP
(Pretty Good
Privacy)

GPG keys

- **Private key**
(yours only)
 - Encrypted with a *passphrase*

- **Public key**
 - Give it to people so they:
 - Can mail you encrypted stuff
 - Can verify stuff came from you



Caveats

- Mail subjects are **NOT ENCRYPTED!**
- From: you@example.com
To: partner@example.com
Subject: Let's overthrow the government

----- BEGIN PGP MESSAGE -----
ASP0DFJQPW9F8TALIRFYW9RFYASJKRFY7S
QWE8R7HF9AW8E7F9AWE88R7JAW99RFA9W7
A0S89F7VH9AW8E7RFH938457FJ9WCFHYKA

- Subject: ...

Keysigning party

- Meet people **in person**
- Exchange **public keys** and/or **fingerprints**
- You can sign their keys later
- Look for the signing-party package!
- You are not qualified to check government-issued IDs :)



- <http://mikegerwitz.com/papers/git-horror-story>
- <http://thehackernews.com/2013/01/hundreds-of-ssh-private-keys-exposed.html>

Are we out of time already?

- **Come to the GPG/Crypto BoF!**
- **<https://wiki.gnome.org/GUADEC/2014/BOFs/Crypto>**
- **Please write your name there!**
- **Federico Mena Quintero <federico@gnome.org>**

GPG fingerprint:

263F 590F 7E0F E1CB 3EA2

74B0 1676 37EB 6FB8 DCCE

GPG, SSH, and Identity for Beginners

GUADEC Strasbourg, July 2014

Federico Mena Quintero

federico@gnome.org

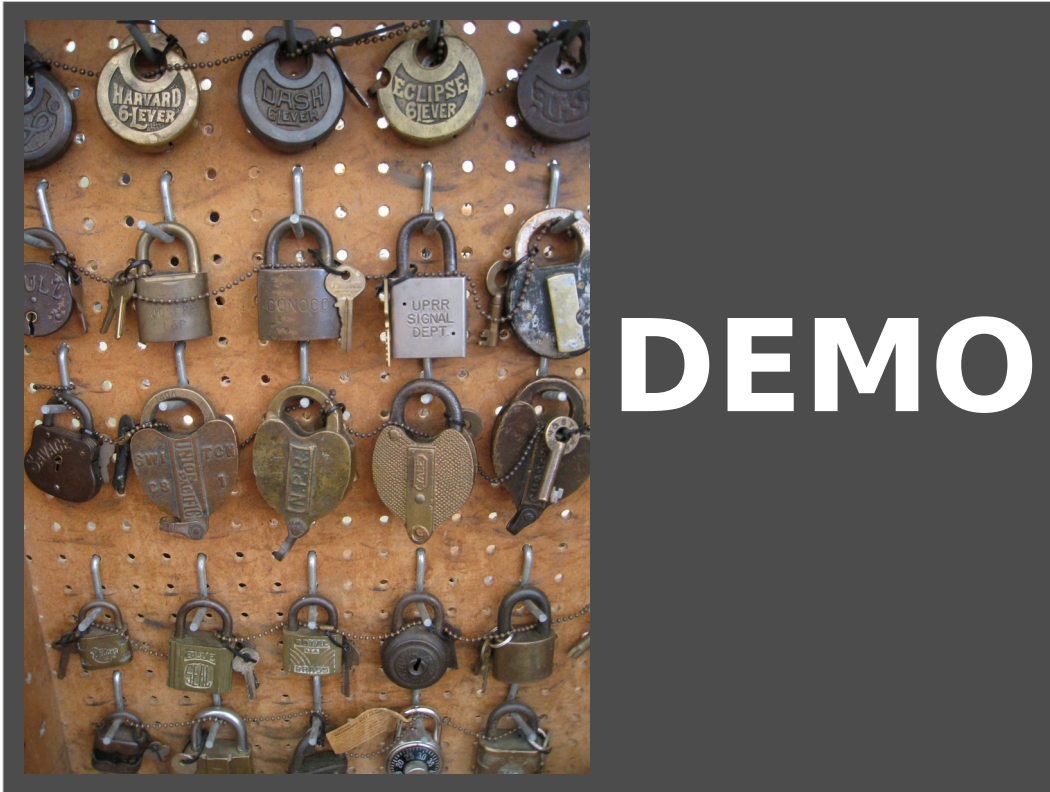
263F 590F 7E0F E1CB 3EA2

74B0 1676 37EB 6FB8 DCCE



I gave this talk at GUADEC, the annual GNOME conference. It was intended for people who are just beginning to learn about public-key cryptography and things like SSH.

Those hexadecimal digits at the bottom are my GPG public key's fingerprint, in case you need to send me confidential information, or more commonly, if you need to ensure that something I signed indeed came from me.



Locks (MyEyeSees) - <https://flic.kr/p/7Xbapi>

First I gave a real-world demo with some physical props: wooden boxes with little padlocks, wax that you melt with a candle and stamps or seals for the wax.

I was the king; volunteers in the audience were my generals. I had their boxes and locks; they had the respective keys. I had my stamp; they had copies of the imprint. Thus I could send messages to them, so that only them could open the boxes, and they could ensure that the messages really came from me.



- **Crypto is **Hard(tm)****
- **Be diligent...**
- **... even if you aren't an expert**
- **It's a prophylactic**

Window cleaner photo (chicagogeek) -
<https://flic.kr/p/8u6uRu>

Cryptography is *really hard* to do properly. It's like juggling eggs while someone is throwing rocks at you.

Now that everything is on the internet, with malware everywhere, and malicious nation-states watching over everyone, all programmers need to know at least the fundamentals of security and cryptography if they are to write competent software.

At some point in the 80s it became accepted to talk about safe sex. We need to make it accepted to talk about privacy and security.

- This is not a talk about **OPSEC**
- **National Operations Security**
- **Spy stuff - “tradecraft”**
- **Useful to covert organizations in general**



White spy (Gord Fynes) - <https://flic.kr/p/4TPxur>

Black spy (Gord Fynes) - <https://flic.kr/p/4TPwYv>

This is not a talk about the kind of procedures you need to follow if you are a covert organization, a criminal, or a government spy. They all need secrecy and secret communications.

However, *they need to stay hidden*. You probably don't; you just need to stay safe, and you need your privacy preserved.

Learning about OPSEC is both revolting and morbidly interesting.

Public-key Cryptography

First we will talk about public-key cryptography, which is what we did with the boxes, locks, seals, and imprints.



- **Symmetric-key cryptography**
- **Easy to understand**
- **Key distribution is a problem**

Decoder ring (Genevieve) - <https://flic.kr/p/bXtoMA>

Everyone learns symmetric-key cryptography when they are children. Substitute letters in the alphabet for the next letter, or for a permutation of the whole alphabet.

This is easy to understand.

Distributing the secret keys (the permutation of the alphabet) is problematic, since you need to be in secure contact with your peer already – so why not give them your message right there in the first place.

- **Private key**
 - The physical key
 - The seal
 - **DO NOT SHARE**



- **Public key**
 - The lock/box
 - The imprint
 - **SHARE**



Numbers (Duncan C.) - <https://flic.kr/p/7XEruh>

Key (Brenda Clarke) - <https://flic.kr/p/6m7iJL>

Box (Brenda Clarke) - <https://flic.kr/p/6hkKQB>

Wax and stamps (Esther Simpson) -

<https://flic.kr/p/rq2Po>

Imprint (Eunice) - <https://flic.kr/p/7piVnt>

Public-key crypto has two parts. The private key is yours only, and you need to keep it secret. In our demo, the private key is the actual physical key to open the locks, and the seals for wax.

The public key is for you to give away to your peers. They don't need to be particularly careful around it. In our demo, the public key is the boxes with locks, and the wax imprints.

Confidentiality

Public-key crypto gives you four things:

1. Confidentiality is keeping the contents of the communications secret. Only the recipient can read the decrypted message.

Data integrity

2. Data integrity means the recipient can verify that the data you sent them has not been tampered while in transit.

Authentication

3. Authentication means the recipient can verify that the message really comes from you and not an impostor.

Non-repudiation

4. Non-repudiation means that you cannot deny that you wrote a particular message.

Sometimes you want this property, and sometimes you don't. It depends on whether you sign your messages or not.

What can we do?

Your private key



Their public key



So, what can we do with public key crypto?

Assume you have generated your key pair (the public and private keys); assume you have distributed your public key to your peers.



Email

We can do email!

Send an email

Encrypt
with their public key
("This is for **you** only")
Lock it in a box



Sign
with your private key
("This comes from **me**")
Stamp it with my seal

If you have someone's public key, you can write encrypted email so that only they will be able to read it. This is equivalent to putting your message in a box with a lock, where only the recipient has the key to the lock.

You can also sign your email with your own private key. This is like stamping a wax seal on the message to prove that it indeed comes from you (and since this is digital, to guarantee that the message is not modified by a malicious third-party.)

Receive an email



Decrypt
with your private key
("Only *I* can read this")
Unlock the box

Authenticate
with their public key
("I'm assured *you*
wrote ***exactly this***")
Check the imprint



When you receive an encrypted email (that was encrypted with your public key), you decrypt it with your private key.

Presumably you also have the public key of your peer. If they signed their message with their private key, you can then use their public key to check that the message is authentic – that *they* wrote it, not an impostor, and that what you received is exactly what they wrote, not a tampered version.

Distributing packages

You can use public-key crypto to validate software packages that you distribute.

Distribute a package



Sign
with your private key
("This comes from *me*")
Stamp it with my seal

You make a tarball (or RPM, or DEB, etc.) and you *sign it* with your private key.

Distribute a package

Let them ensure
this comes from you

Let them validate
the tarball's integrity



Your announcement email:

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1  
  
Hi everyone!  
  
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbdf9efceab68c746.  
  
Sincerely,  
  J. Random Hacker  
  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1  
  
iQGVawUBU45IaJEDl9iNKTGaAQKcAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

Sign
with your private key
("This comes from me")
Stamp it with my seal

When you distribute your package, you can publish a signed announcement that includes a cryptographic hash of your package, so people can validate the package's contents and its origin.

Receive a package

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1  
  
Hi everyone!  
  
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcd6f9efceab68c746.  
  
Sincerely,  
  J. Random Hacker  
  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1  
  
iQGVAwUBU45IaJEDl9iNKTGaAQKcAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

And when you receive a package and/or its announcement...

Receive a package

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1  
  
Hi everyone!  
  
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbd6f9efceab68c746.  
  
Sincerely,  
  J. Random Hacker  
  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1  
  
iQGVAwUBU45IaJEDl9iNKTGaAQKcAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

**Authenticate
with their public key
("I'm assured *you*
packaged *exactly this*")
Check the imprint**



... you validate it against the public key you have from the package's author.

Receive a package

```
-----BEGIN PGP SIGNED MESSAGE-----  
Hash: SHA1
```

```
Hi everyone!
```

```
I've just released foobar-3.4.5.tar.xz.  
The SHA256 hash is af8abcbd6f9efceab68c746.
```

```
Sincerely,  
J. Random Hacker
```

```
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1
```

```
iQGVAwUBU45IaJEDl9iNKTGaAQKcAv8CFVTBvbN0u  
...  
-----END PGP SIGNATURE-----
```

Your email client checks the signature

You check the tarball's hash

**Authenticate
with their public key
("I'm assured *you*
packaged *exactly this*")
Check the imprint**



Some people send out signed emails with package announcements, that include a cryptographic hash of the package's binary.

Other people publish the GPG signatures of the package binaries along with the packages themselves.

These are equivalent.

Generating key-pairs

Your private key



Your public key



Generating keys is not hard. You tell GPG to do it, or you use a graphical front-end like Seahorse.

Generating key-pairs

Your private key

Your public key



Numbers (Duncan C) - <https://flic.kr/p/7XEruh>

A keypair is essentially a pair of big random numbers. You keep the private one really secret, and you can publish the public one.

Distributing **public** keys

How do you
give these
to people?



But how do you get your public key out to people that you'll communicate with?

Receiving **public** keys

How do you
find/receive
these?



How do you
ensure the
keys are
authentic?



And how do you obtain other people's public keys so that you can validate messages that come from them?

How do you ensure that keys are authentic? If you look for Ed Snowden's keys online, you'll find fake ones.

Trust

Let's talk about trust.

Trust models

Or rather, trust models.

This is not “do I trust this person with my life's secrets”, but rather, “do I trust this person to be who they say they are?”.

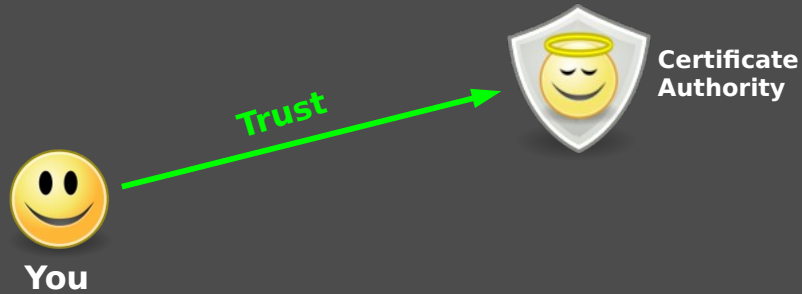
Centralized model (Certificate Authorities)



You

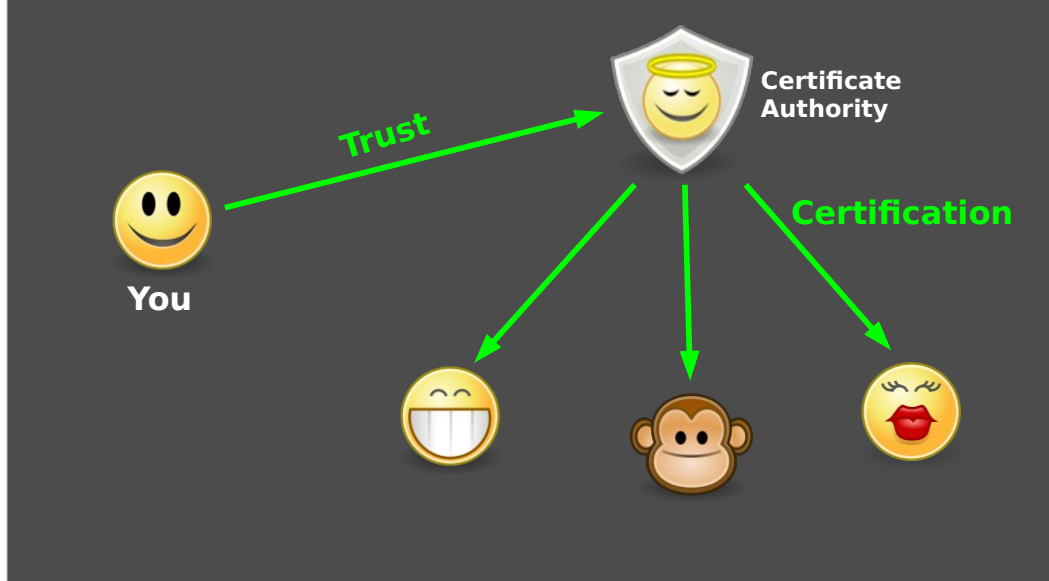
The web works with a centralized trust model. Certificate Authorities (CAs) are entities which validate people's (or companies') identities.

Centralized model (Certificate Authorities)



Your web browser is preconfigured to trust certificate authorities...

Centralized model (Certificate Authorities)

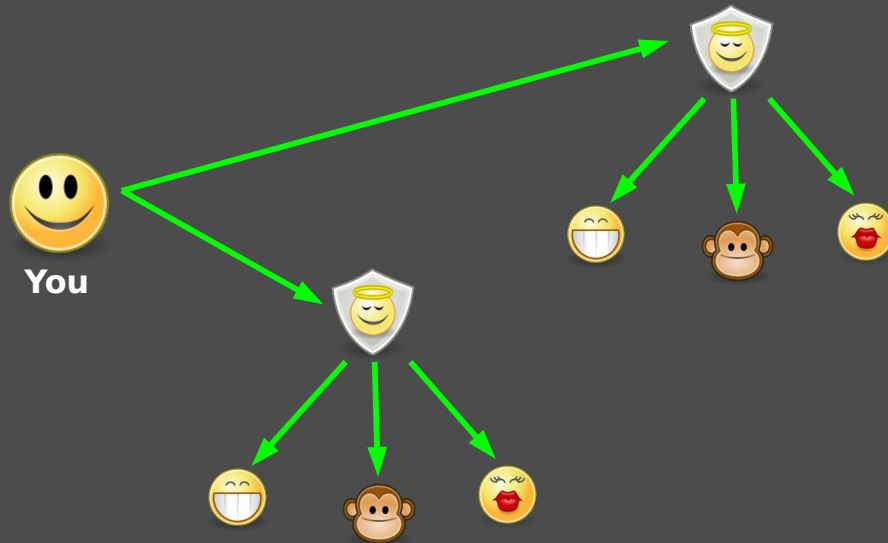


... which in turn validate other entities.

This means, “I trust the certificate authority to have done its homework in ensuring that those people are who they say they are”.

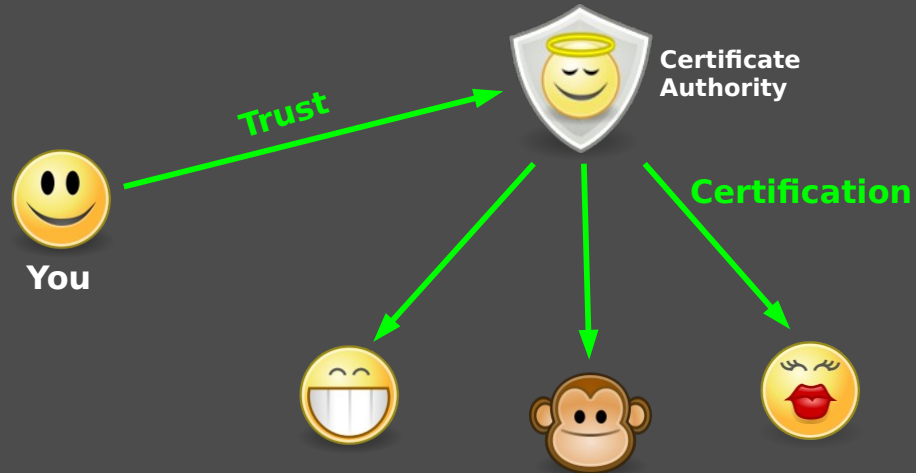
An impostor won't be validated by the certificate authority; if you try to connect to an impostor website, your web browser will warn you.

Centralized model (Certificate Authorities)



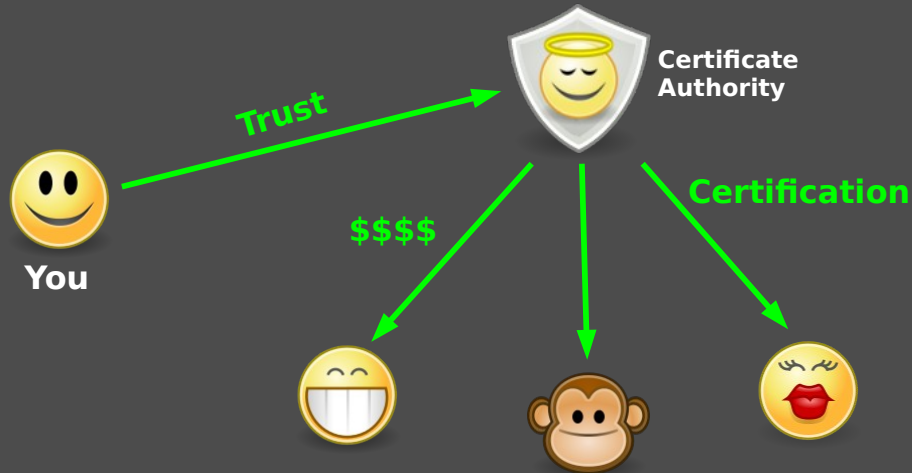
There are multiple CAs, each with their trustees.

Centralized model (Certificate Authorities)



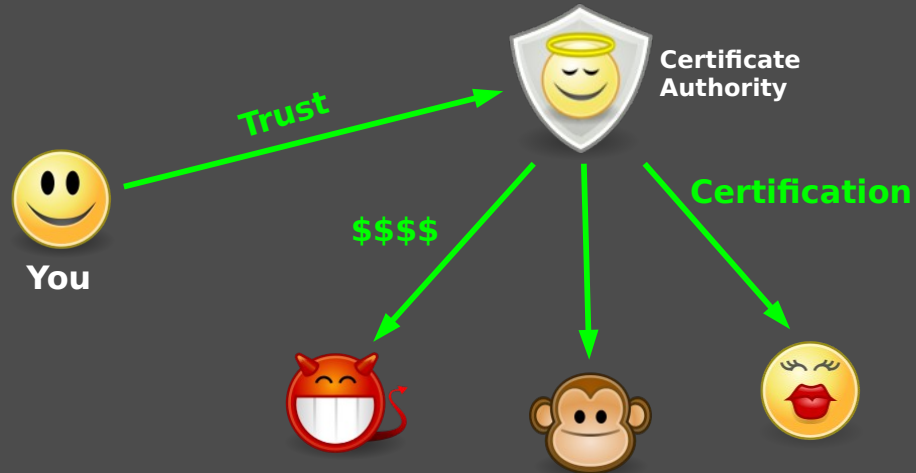
The problem is...

Centralized model (Certificate Authorities)



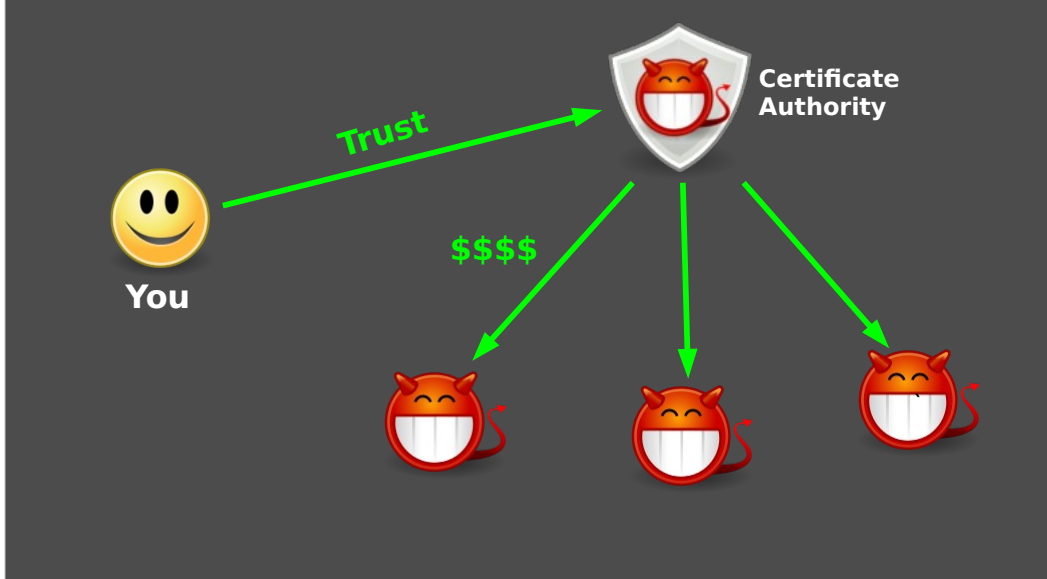
... that money is involved. People *pay* certificate authorities. You don't know if the CA actually does its homework to validate those people.

Centralized model (Certificate Authorities)



So, there are cases where “good” CAs inadvertently validate a rogue entity...

Centralized model (Certificate Authorities)



... or where malicious CAs validate rogue entities.

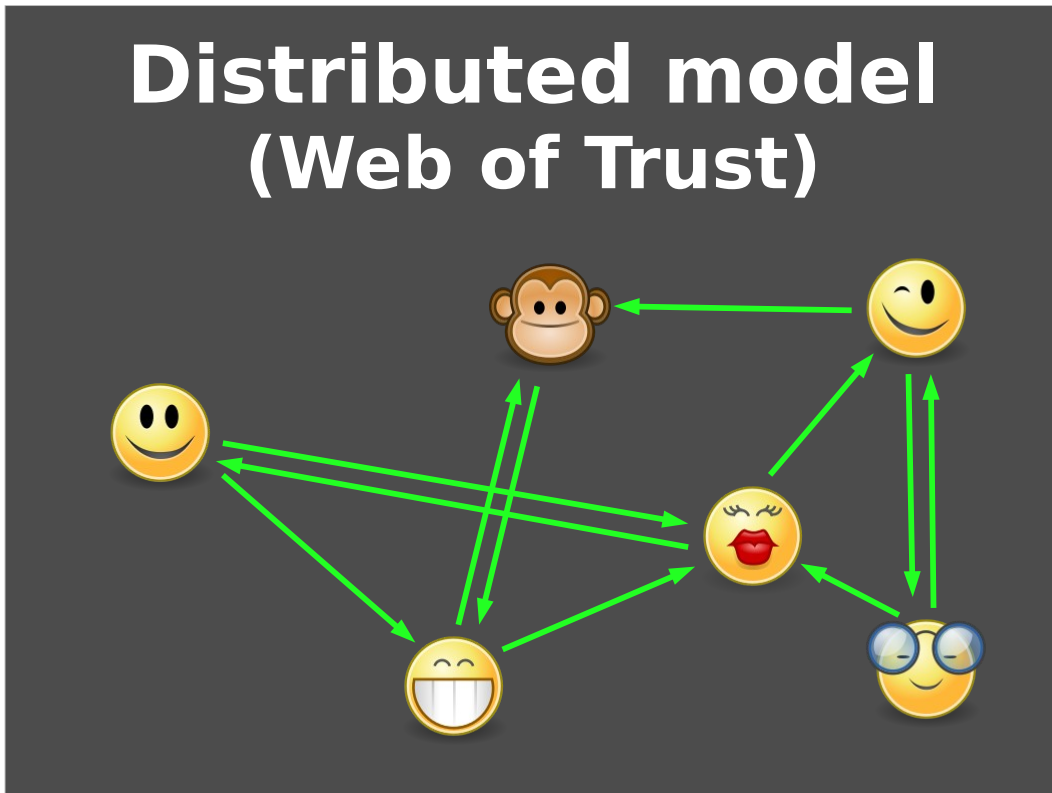
Malicious nation-states do this all the time to spy on people's web-browsing habits.



You

Sad you, as a result.

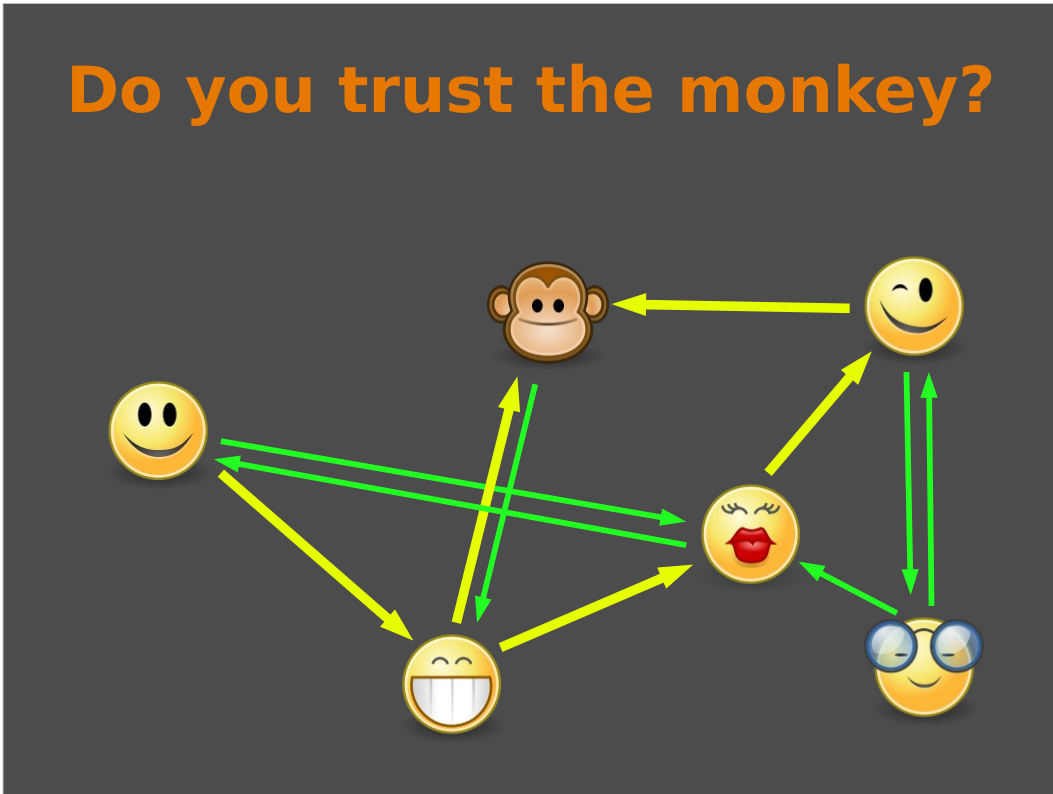
Distributed model (Web of Trust)



GPG uses a distributed trust model, the “web of trust”.

Here, individuals say whom they trust to be genuine. Or technically, whose keys they have made sure to actually belong to the peers they think they'll be communicating with.

Do you trust the monkey?



Again, “do you trust the monkey” doesn't mean that you'll give it your house keys or your life's secrets. It means that you have ensured that the monkey is the monkey you think it is, and none other.

If you haven't received the monkey's public keys directly from him, you need to trust your friends. The more friends you have validated, who in turn have validated the monkey, the more sure you can be that the monkey is genuine and not a fake monkey.

SSH (secure shell)

SSH, our beloved software to connect to remote computers, also uses public-key crypto.

```
$ telnet somehost.example.com
Trying 192.168.1.87...

somehost login: _
```

SSH was written to replace Telnet. Telnet also lets you connect to other computers, but it is unencrypted.

Back in the university, we would run TCP sniffers on the network and learn our classmates' passwords, and what they were typing. Not good. SSH prevents that.

```
$ telnet somehost.example.com
Trying 192.168.1.87...

somehost login: federico
Password: _
```

Passwords in cleartext = BAD


```
$ telnet somehost.example.com  
Trying 192.168.1.87...
```

```
somehost login: federico  
Password: _
```

KABOOM

REALLY BAD

SSH user (you)

- **Private key**
(yours only)
 - Encrypted with a *passphrase*



- **Public key**
 - Copied to
~/.ssh/authorized_keys
in destination
machines



When you create an SSH keypair, you keep your private key in your own home directory (and you must keep it secret). SSH takes the precaution of encrypting your key with a *passphrase*, so that if your key gets stolen, the attacker will at least have a hard time trying to decrypt the key.

You are supposed to copy your public key by hand to destination machines. Sometimes a sysadmin can do that for you.

SSH server (git.gnome.org)

- **Private key**
(server's only)
- **Public key**
 - Copied to `~/.ssh/known_hosts` in your machine, automatically, when you first ssh in

Each machine to which you connect (“a server”) has its own keypair.

The private key of course stays secret. It gets generated when the sysadmin installs SSH.

SSH takes care of copying the server's public key to your home directory when you first connect to the server. This is so that in the future you'll be warned if the server's identity changes or if there is a man-in-the-middle attack.

Am I connecting to the correct machine?

```
pambazo:~$ ssh tlacoyo.local
The authenticity of host 'tlacoyo.local (192.168.1.87)' can't be
established.
ECDSA key fingerprint is
fc:26:07:61:c9:2c:1f:6c:90:64:59:d7:11:6d:6f:06.
Are you sure you want to continue connecting (yes/no)? _
```

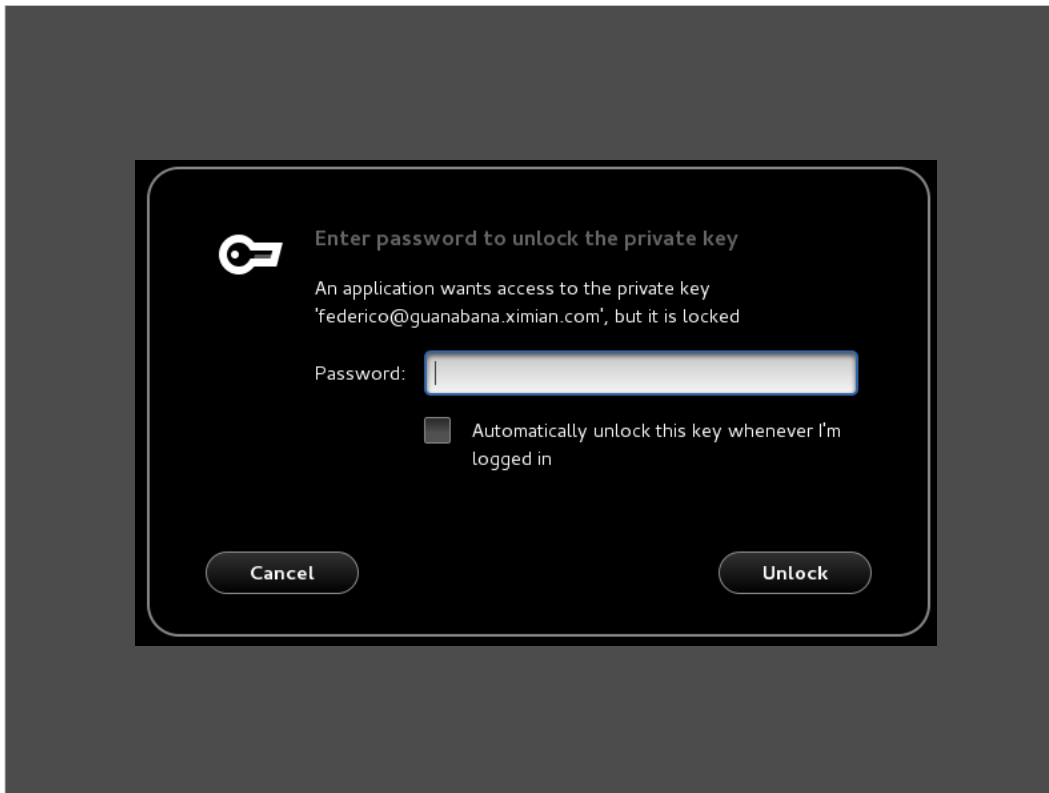
Ask your sysadmin about this number

Hint:

```
tlacoyo:~$ ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key
256 fc:26:07:61:c9:2c:1f:6c:90:64:59:d7:11:6d:6f:06 root@linux-szu1 (ECDSA)
```

If SSH doesn't know a server (like the first time you connect to it), it will give you the server's key's fingerprint. You are supposed to validate that fingerprint!

Ask the server's sysadmin for confirmation.



Also, GNOME will prompt you for your SSH passphrase.



Again, this is so that SSH can decrypt your private key and use the result to encrypt the rest of your session.

(That's a simplification of how things actually work, but it's good enough.)

```
pambazo:~/src$ git clone ssh://git.gnome.org/git/seahorse
Cloning into 'seahorse'...
The authenticity of host 'git.gnome.org (209.132.180.184)' can't be
established.
RSA key fingerprint is
00:39:fd:1a:a4:2c:6b:28:b8:2e:95:31:c2:90:72:03.
Are you sure you want to continue connecting (yes/no)? _
```

Do we publish this anywhere?

I haven't seen any GNOME web pages where we advertise the fingerprints of our servers.

One of the sysadmins told me that SSH can automatically validate this through secure DNS. I don't understand how that works, so I can't explain it here.

GPG (GNU Privacy Guard)

GPG stands for GNU Privacy Guard...

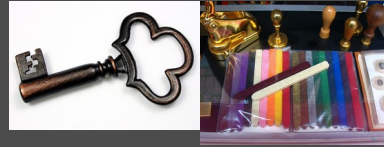
PGP **(Pretty Good** **Privacy)**

... which is a play on the name of the software from which it was originally derived.

GPG keys

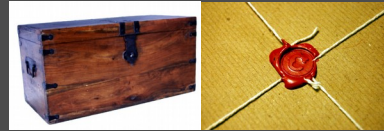
- **Private key**
(yours only)

- Encrypted with a *passphrase*



- **Public key**

- Give it to people so they:
 - Can mail you encrypted stuff
 - Can verify stuff came from you



Reminder of how GPG keypairs work.

Again, your private key must stay secret. Like SSH, GPG keeps your private key encrypted with a passphrase, so that if someone steals your laptop with your keys, they'll have a hard time impersonating you or reading encrypted mail sent to you.

Caveats

- Mail subjects are **NOT ENCRYPTED!**
- From: you@example.com
To: partner@example.com
Subject: Let's overthrow the government

```
----- BEGIN PGP MESSAGE -----  
ASPODFJQPW9F8TALIRFYW9RFYASJKRFY7S  
QWE8R7HF9AW8E7F9AWE88R7JAW99RFA9W7  
A0S89F7VH9AW8E7RFH938457FJ9WCFHYKA
```

- Subject: ...

GPG email sucks in various ways.

One thing to keep in mind is that email subjects are *not encrypted*; only the mail's body is. So, don't put stuff in the subject that is meant to be secret. Use a subject like “...” that doesn't give information to an eavesdropper.

Keysigning party

- Meet people **in person**
- Exchange **public keys** and/or **fingerprints**
- You can sign their keys later
- Look for the signing-party package!
- You are not qualified to check government-issued IDs :)

You can have a keysigning party to obtain a bunch of people's public keys.

You can exchange the public keys themselves, or just the fingerprints if you can obtain the keys later through public key servers.

There is a meme that you need to check government-issued IDs for the people in keysigning parties. This is bullshit. IDs are easy to forge, and you don't know what a valid passport from the other side of the world looks like, anyway.

In general, only accept keys from people you have known personally for some time.



- <http://mikegerwitz.com/papers/git-horror-story>
- <http://thehackernews.com/2013/01/hundreds-of-ssh-private-keys-exposed.html>

Some horror stories for your amusement and education.

Are we out of time already?

- **Come to the GPG/Crypto BoF!**
- **<https://wiki.gnome.org/GUADEC/2014/BOFs/Crypto>**
- **Please write your name there!**
- **Federico Mena Quintero <federico@gnome.org>**

**GPG fingerprint:
263F 590F 7E0F E1CB 3EA2
74B0 1676 37EB 6FB8 DCCE**

At GUADEC we also held a Birds-of-a-Feather session to get people started with GPG.

I gave out USB sticks, courtesy of SUSE, so that people could have their very own encrypted USB sticks to carry around sensitive data.

Then we created GPG keypairs for people, and showed how to keep a master private key in the encrypted USB, while your laptop only carries secondary keys for signing/encryption.