Gluing Together Desktop Crypto

Stef Walter

Stef Walter.

I work for Collabora.

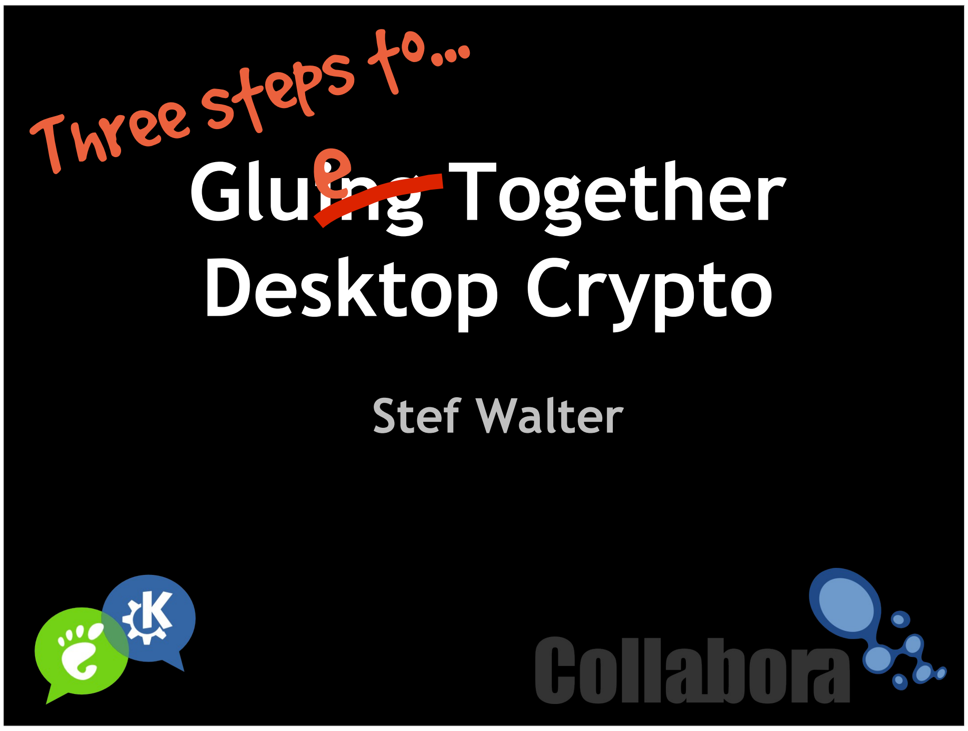
Maintainer of gnome-keyring and seahorse developer.

Interested in making crypto usable.

There are all sorts of cool new ideas for identity and trust and using crypto in inovative and powerful ways. But in order to do that, we need to get our house in order.

If any of this seems mundane, it's because it is. By getting this plumbing to be solid, we can then move onto more interesting and awesome stuff.

Because all cool presentations have the number three in the title, I couldn't resist joining in the fun.



Three steps to...

~~Gluing~~ Glue Together Desktop Crypto

Stef Walter

Three steps to Glue Together Desktop Crypto.

Many technically excellent SSL and crypto libraries on the desktop, but we need to provide glue to use them together.

In some ways gnome-keyring is glue between things like SSH, PGP, and other bits. But this is taking things further.

The user doesn't need to care what library is being used, or which file formats keys are in, and blah blah.

A couple of approaches to solving this problem are currently underway.

Fedora is standardizing on using NSS everywhere, porting all software to use it, and using a central certificate database.

A second approach, which I'll talk to you about today is also underway. It allows more flexibility, but interoperates perfectly with the Fedora approach as well.

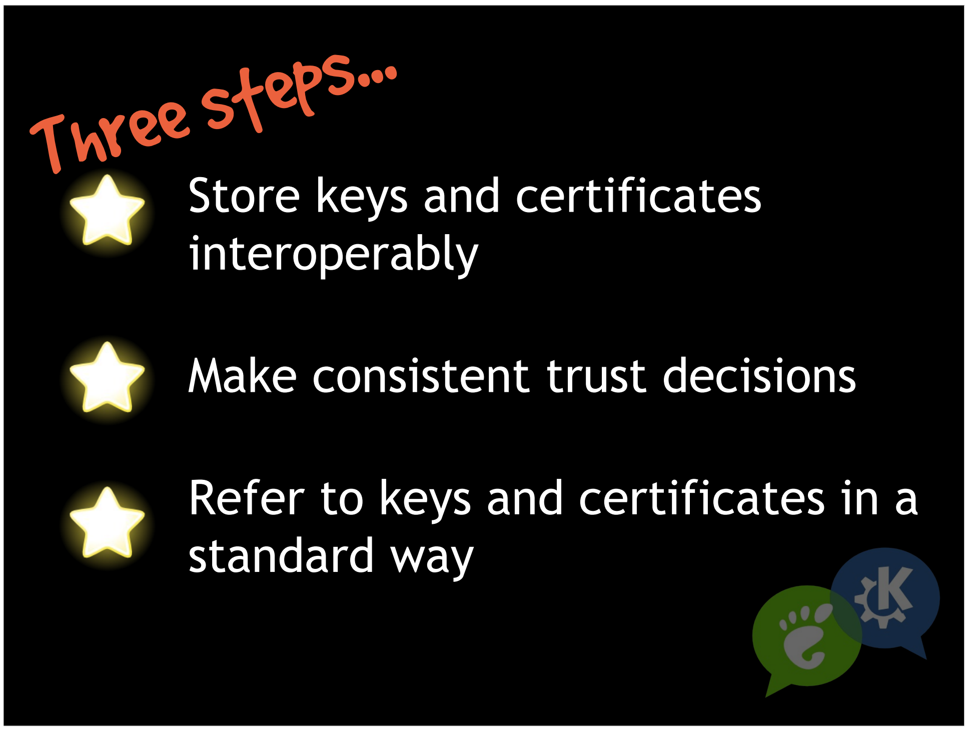
You're going to hear more about what PKCS#11 is and how it can change your life :P

Benefits

 * Flexible for distros, Desktops and sysadmins to match to their needs.

 * Celebrate diversity

 * Integrates really well with smart cards and hardware crypto.



Three steps...

- Store keys and certificates interoperably
- Make consistent trust decisions
- Refer to keys and certificates in a standard way

Store keys and certificates in a way that is interoperable and multiple libraries and components can use them.

Make consistent trust decisions, so that the user doesn't have to try and figure out why one application 'works' while another doesn't, when doing the same thing.

Ability to for one application to refer to a key in a standard way, when it tells another library or application about it.

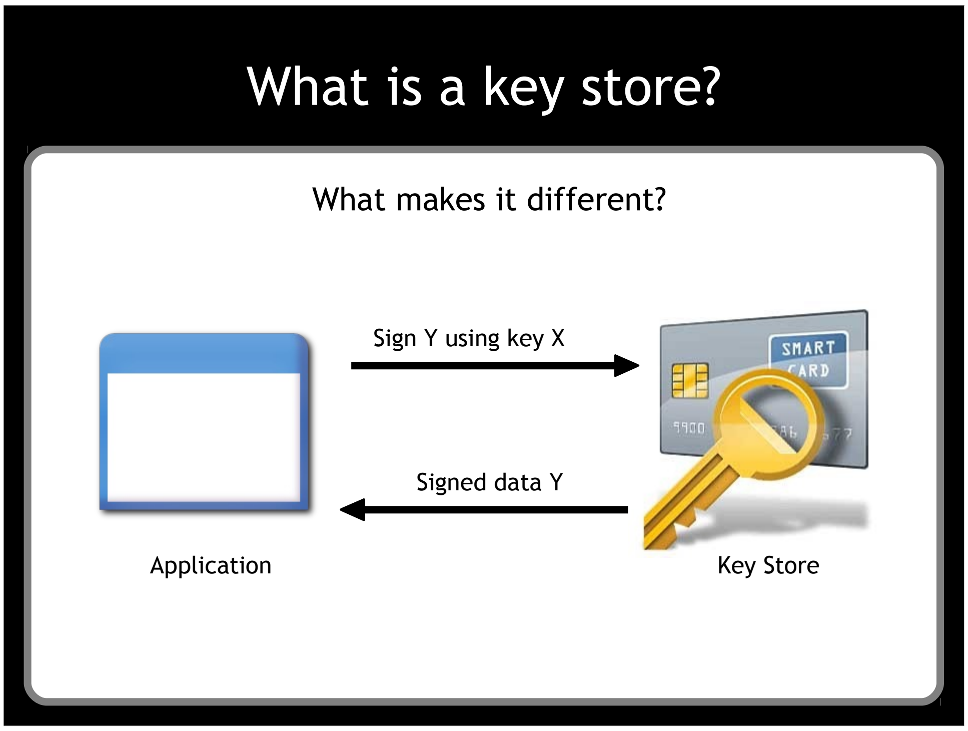
Because we'll be playing with glue today, first a word of warning from Spock.

Don't want to end up like him.

Key Storage

What makes key storage special?

Why not use a file or database or configuration system to store keys.

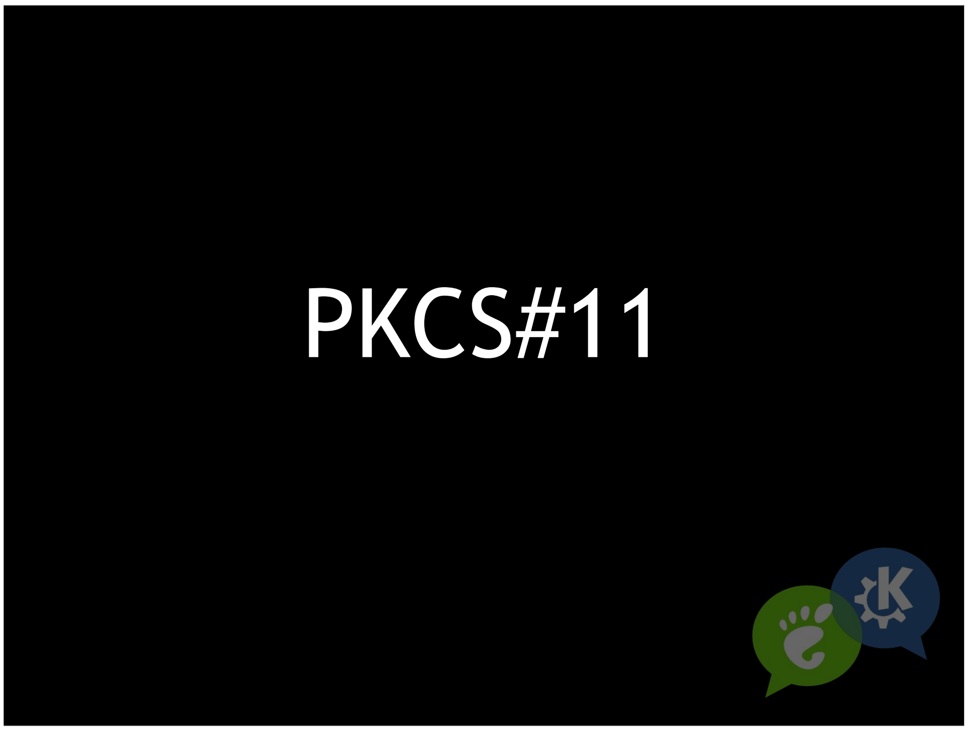


What is a key store?

What makes it different?

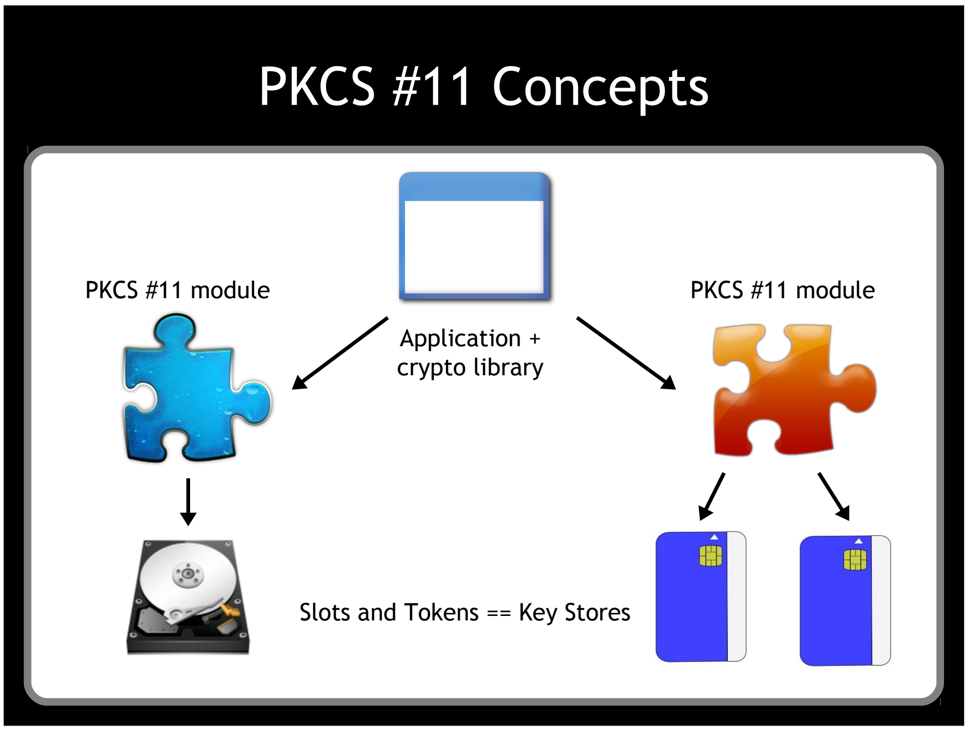
Perform crypto operations using the keys, without the keys leaving the store.

A: The way that keys don't leave the store, and the store performs the operations. Example, smart card.



PKCS#11

This is where PKCS#11 enters the picture.
A standard for key storage (and a few other things).


PKCS #11 Concepts

Common ground between applications and libraries.

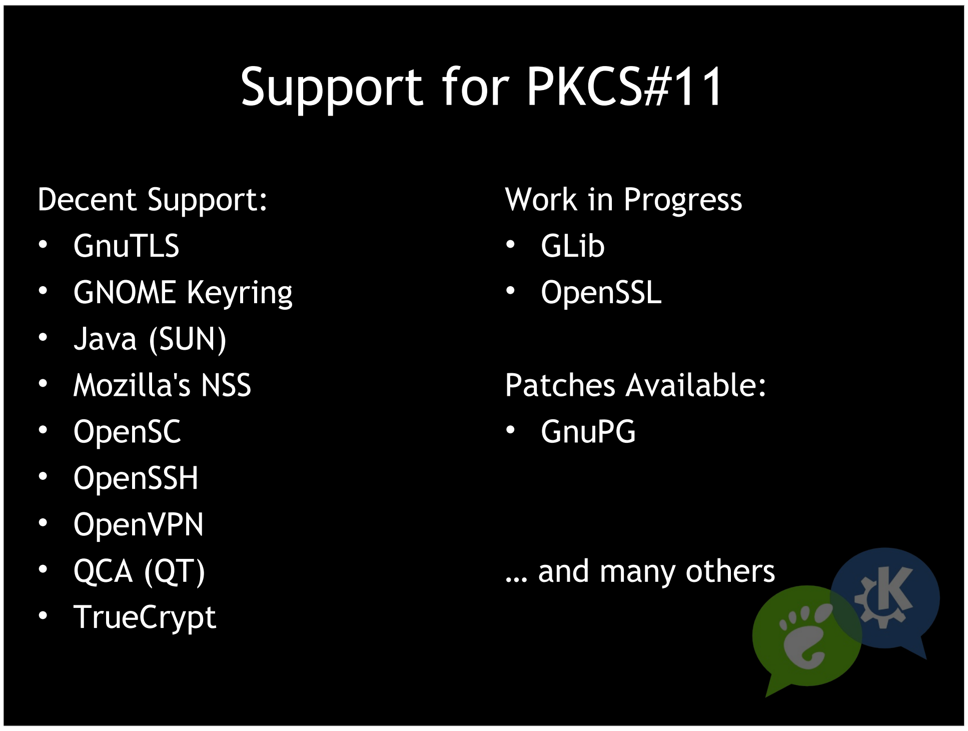
It's defined as a bare bones set of C function pointers.

Specifies application programming interface ("Cryptoki") in C

But more importantly it defines how to do crypto operations on keys without the keys leaving the storage.

An application loads one or more PKCS#11 modules, each of which allow access to a different kind of key storage.

If you have a way of storing keys on your Desktop or application today, you could put together a PKCS#11 module so that other crypto libraries can access those keys.



Support for PKCS#11

Decent Support:

- GnuTLS
- GNOME Keyring
- Java (SUN)
- Mozilla's NSS
- OpenSC
- OpenSSH
- OpenVPN
- QCA (QT)
- TrueCrypt

Work in Progress

- GLib
- OpenSSL

Patches Available:

- GnuPG

… and many others

The main benefit of PKCS#11 is that it's supported, which is why we can use it as glue. The API is older and baroque, but we can live with that.

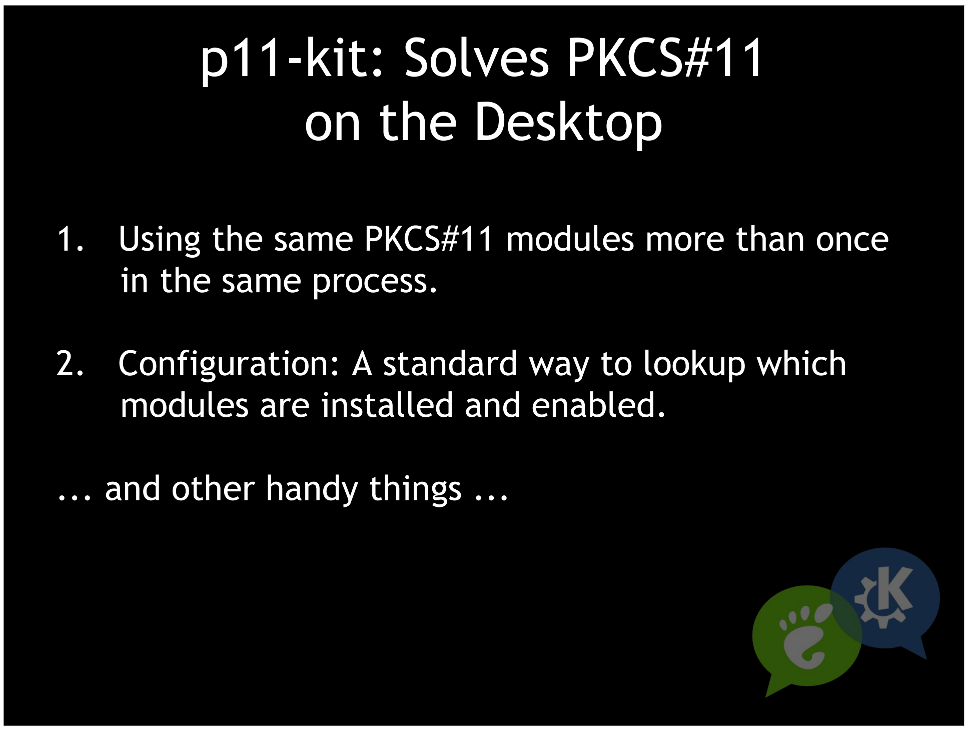
- Support for PKCS#11 in various libraries.

- NSS has great complete support for PKCS#11 and is built around it.

- OpenSSL has a pluggable engine for PKCS#11.

- Gnome Keyring is built around PKCS#11. Even the SSH agent is built around PKCS#11.

- Widely supported.



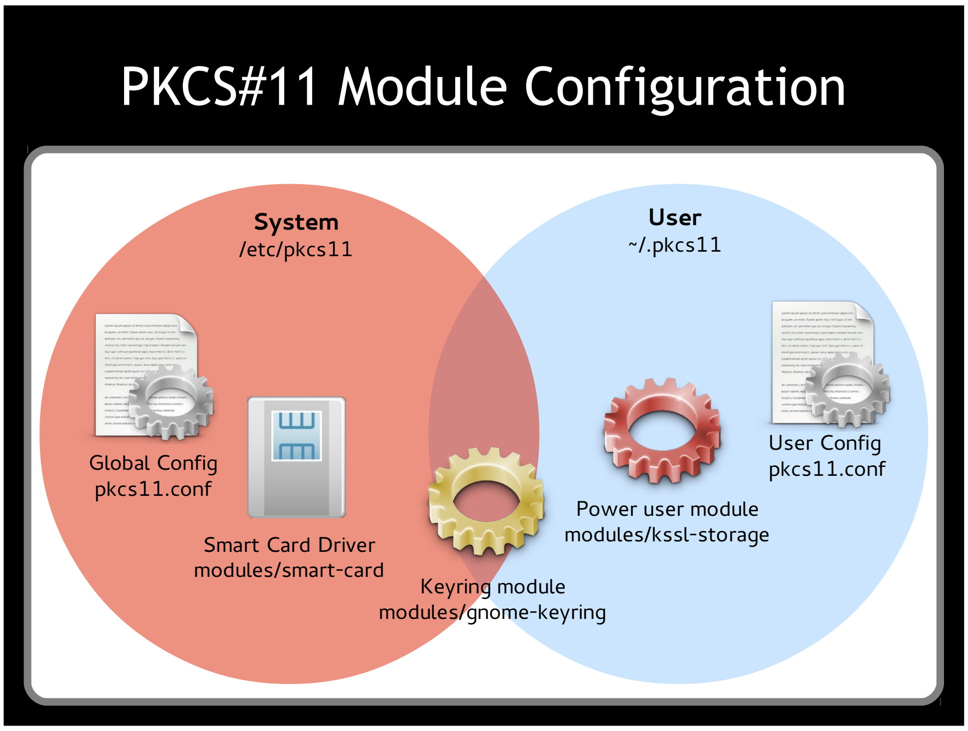
p11-kit: Solves PKCS#11 on the Desktop

1. Using the same PKCS#11 modules more than once in the same process.
2. Configuration: A standard way to lookup which modules are installed and enabled.

... and other handy things ...

(Read the items so I'm not talking over people's reading of them.)

“No brandcuffs?” College Humor



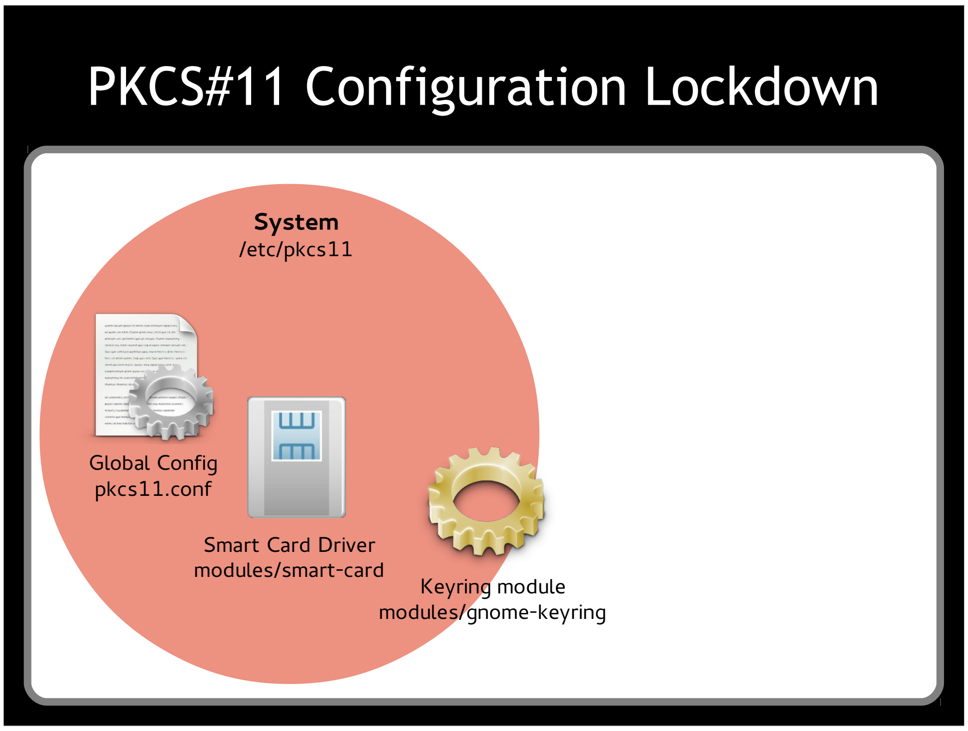
PKCS#11 Module Configuration

Poorly made graphic showing the configuration setup.

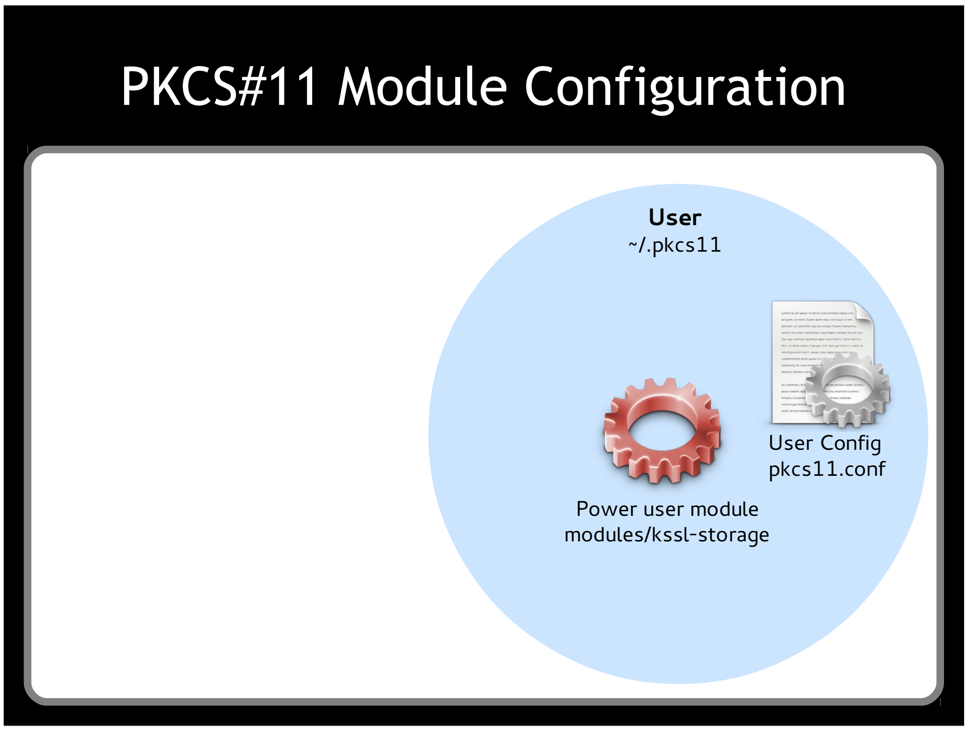
Learned from systemd, pam, NSS.

System and user section, global and user config.

Modules are separate files, and can be installed/updated independently by packages, scripts etc.

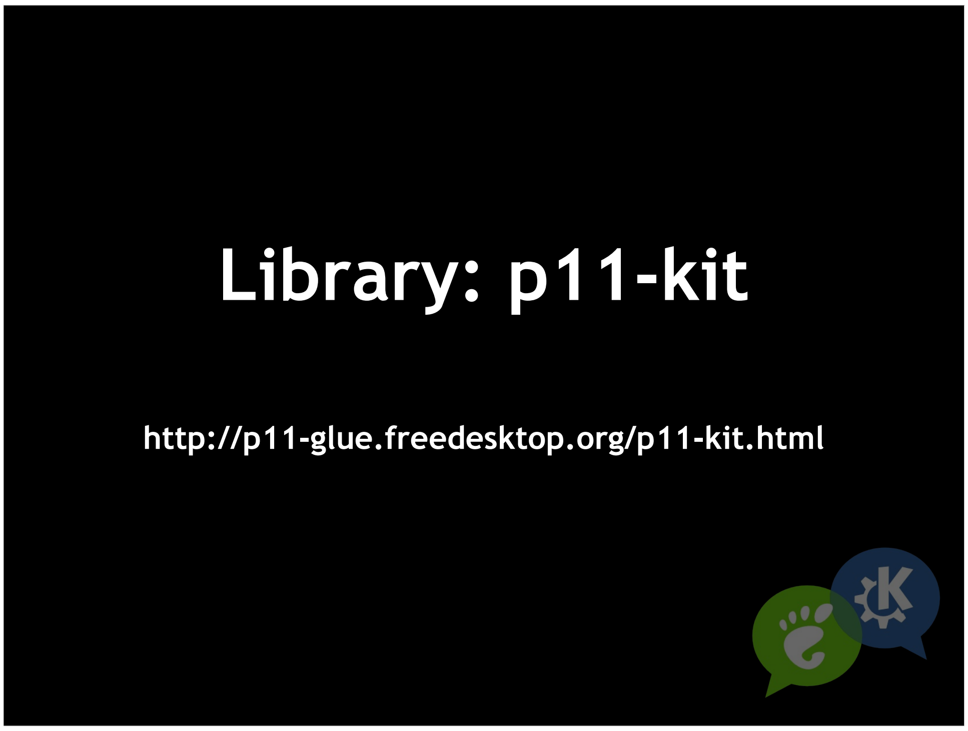

PKCS#11 Configuration Lockdown

Can optionally refuse to load user configuration: ie: lockdown style



PKCS#11 Module Configuration

Can optionally refuse to load system config ie: Developer or power-user mode



Library: p11-kit

http://p11-glue.freedesktop.org/p11-kit.html

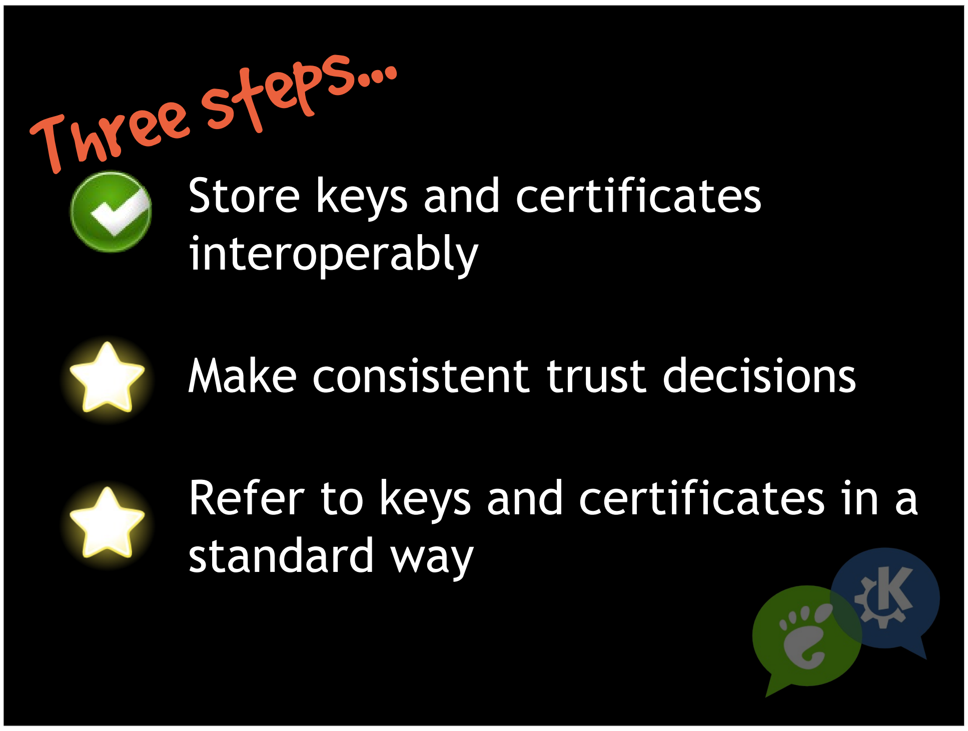
All of this and more is in the little library called p11-kit.

BSD licensed, no dependencies, and full of goodness.

Also has a proxy module, which can work with any application using PKCS#11.
So no code changes necessary to use this goodness.

On freedesktop.org

Integrated into gnutls, gnome-keyring, and soon Glib.



Three steps...

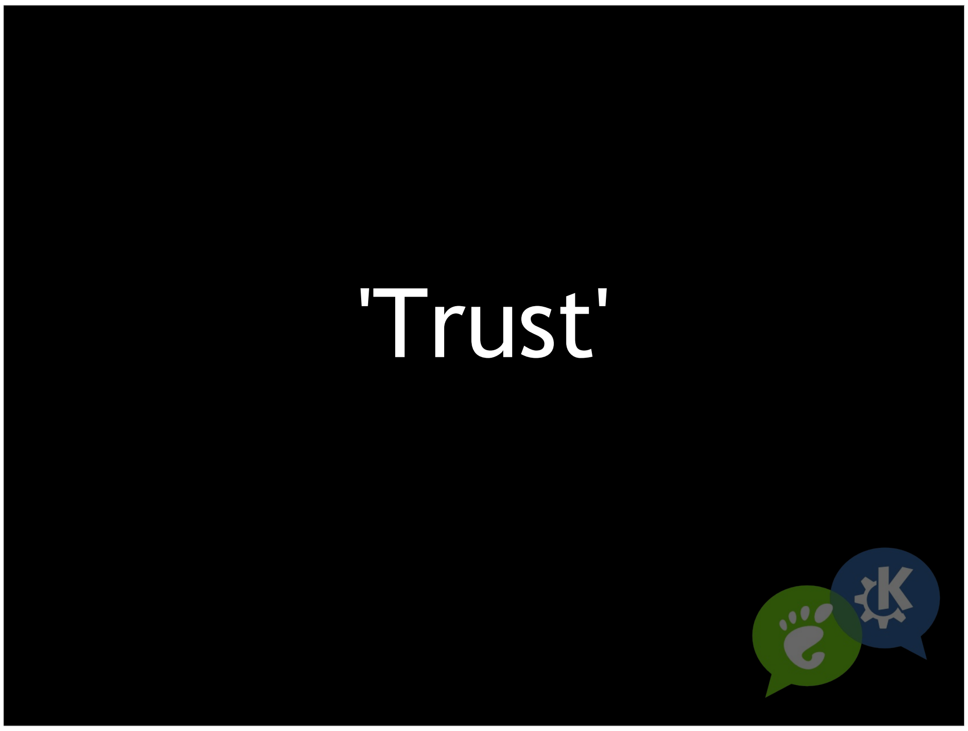
✓ Store keys and certificates interoperably

☆ Make consistent trust decisions

☆ Refer to keys and certificates in a standard way

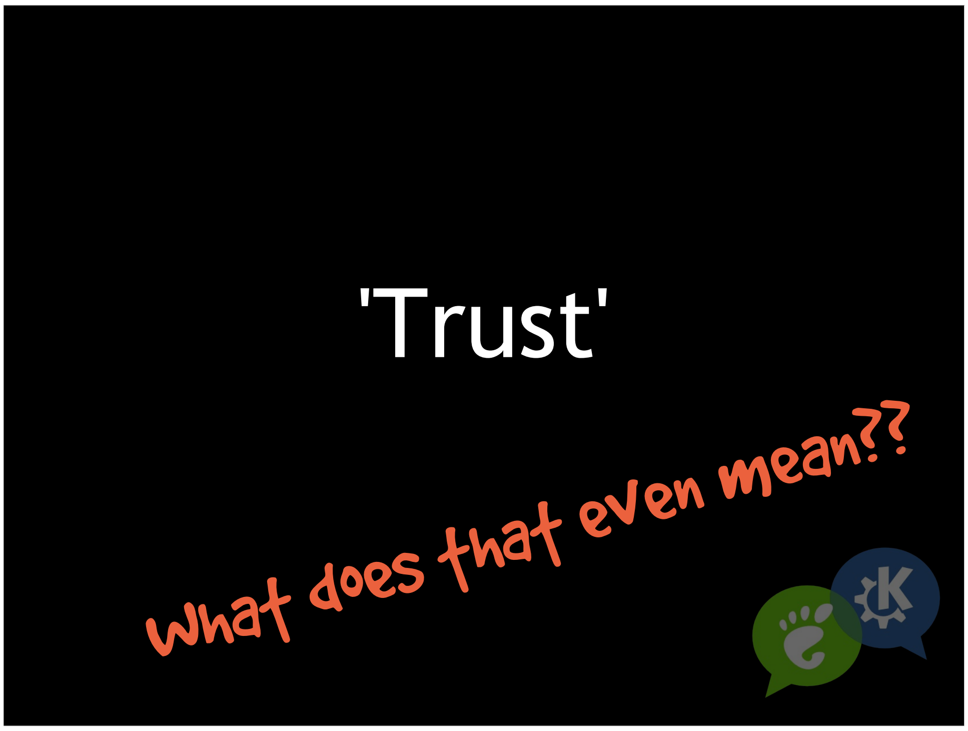
Covered step one, we figured out how to store keys and certs interoperably: Use PKCS#11, and use p11-kit.

Now: Make consistent trust decisions



'Trust'

Trust is a very ambiguous overloaded term.

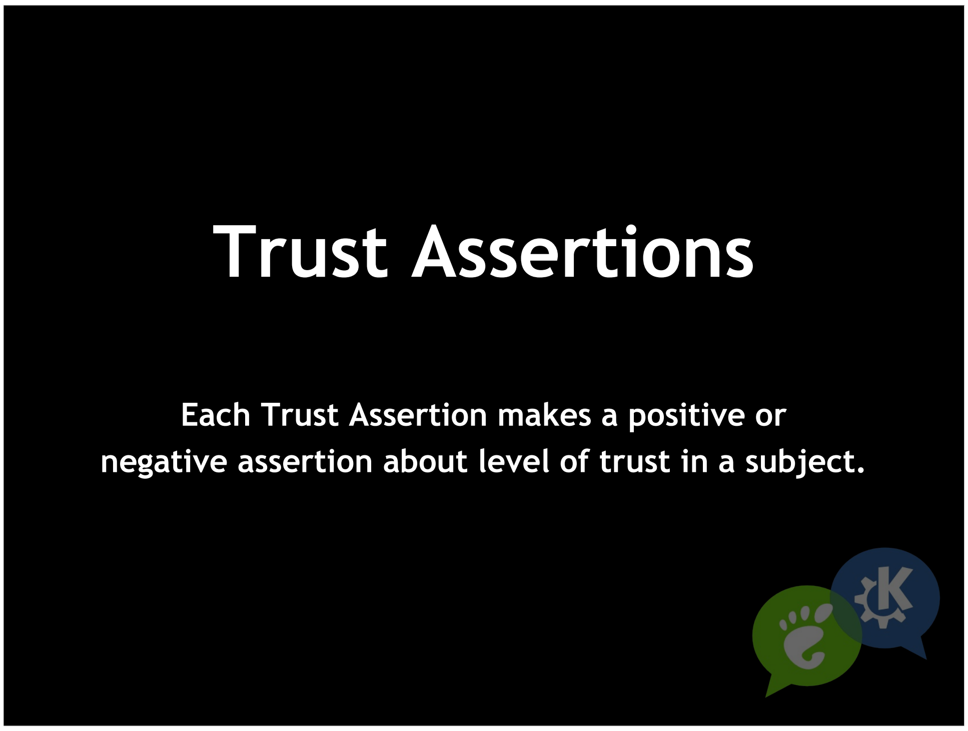


'Trust'

What does that even mean??

If you say the word 'trust' in a room full of developers like this, you're going to have at least as many definitions of the word as there are developers.

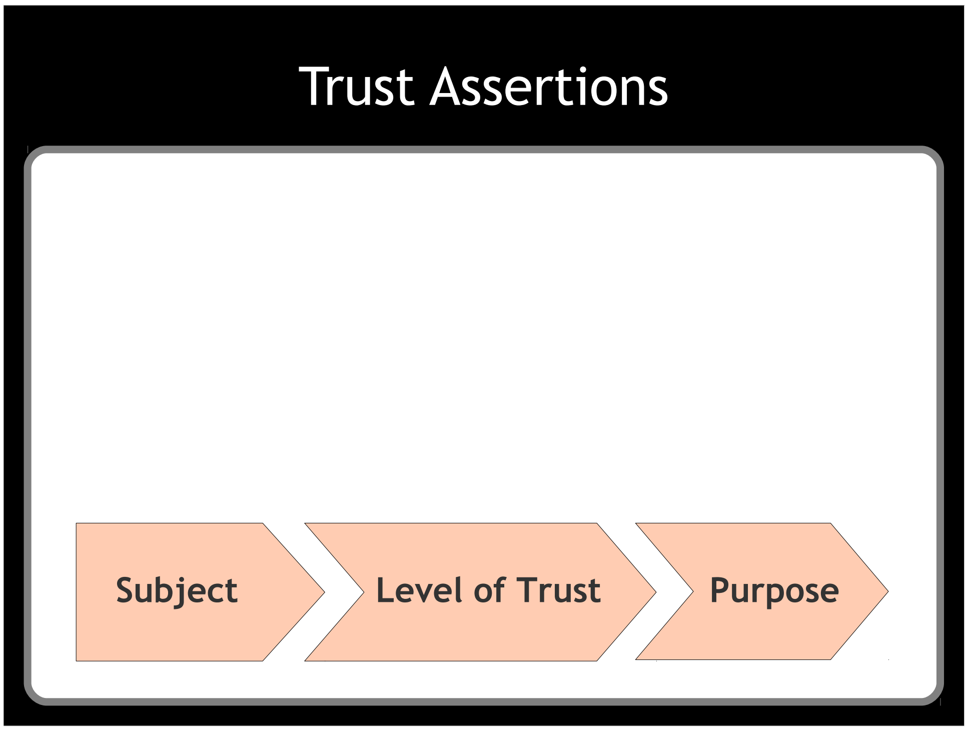
But forget about all those definitions for a second, and let me introduce you to a very specific concept.



Trust Assertions

Each Trust Assertion makes a positive or negative assertion about level of trust in a subject.

See above



Trust Assertions

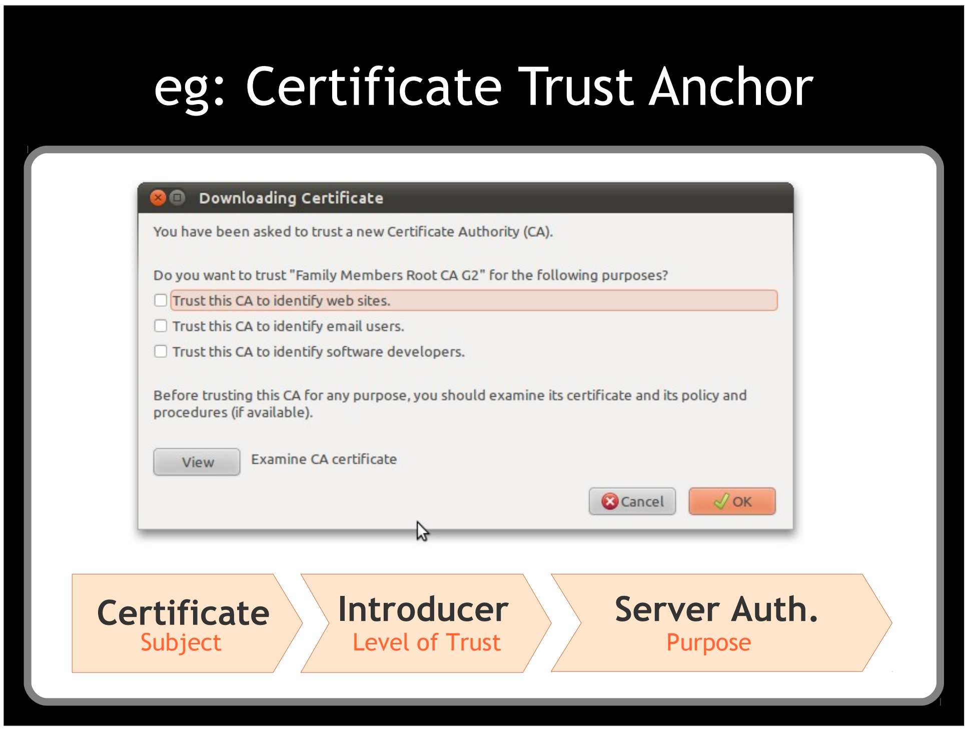
A trust assertion is a tuple containing: a subject, a level of trust, and a purpose.

Subject are things like a certificate, key, fingerprint or signature.

Level of trust: distrust, positive trust, introducer.

All trust is almost always for a given purpose.

You trust your bank with your money, and your school with your kids, but not necessarily vice versa.



eg: Certificate Trust Anchor

Downloading Certificate

You have been asked to trust a new Certificate Authority (CA).

Do you want to trust "Family Members Root CA G2" for the following purposes?

- [ ] Trust this CA to identify web sites.
- [ ] Trust this CA to identify email users.
- [ ] Trust this CA to identify software developers.

Before trusting this CA for any purpose, you should examine its certificate and its policy and procedures (if available).

Example of a trust assertion.

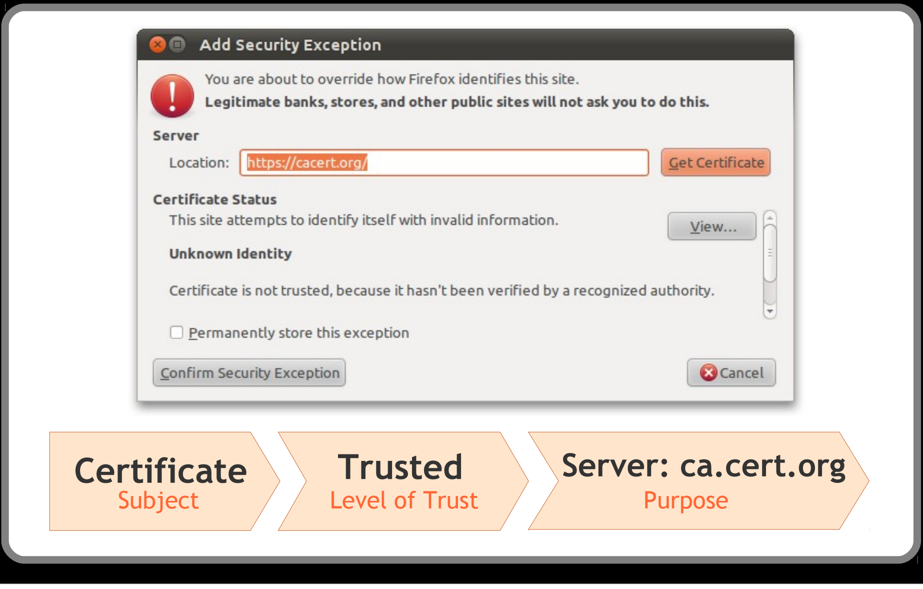
A positive trust assertion.

Certificate is the subject.

Level of trust is introducer

Purpose is server authentication.

eg: Pinned Certificate Exception


Add Security Exception

You are about to override how Firefox identifies this site.

Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://cacert.org/ Get Certificate

Certificate Status

This site attempts to identify itself with invalid information. View...

Unknown Identity

Certificate is not trusted, because it hasn't been verified by a recognized authority.

☐ Permanently store this exception

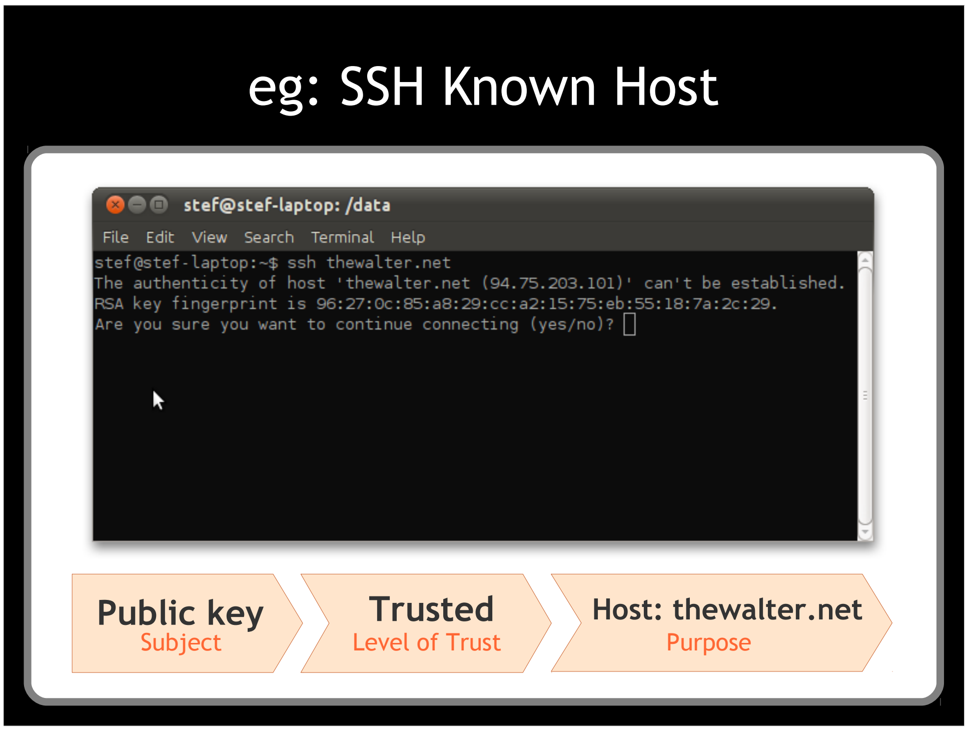
Confirm Security Exception Cancel

Another example

A positive trust assertion.

A pinned certificate exception, used with a specific host (or peer).

Level of trust is: Trusted.



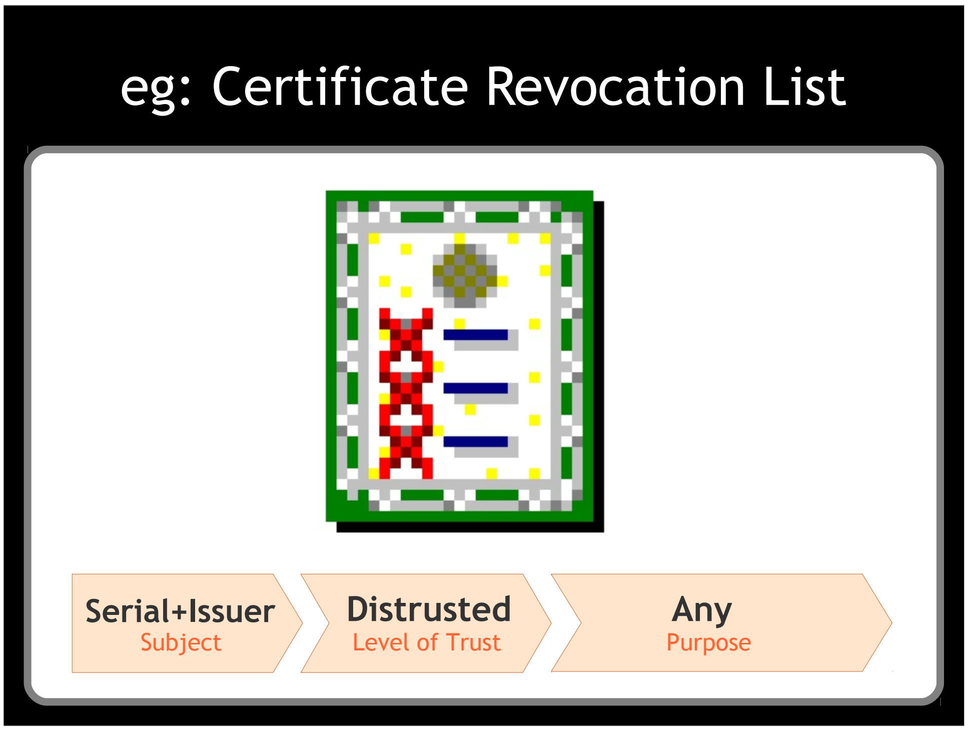
eg: SSH Known Host

```
stef@stef-laptop:~$ ssh thewalter.net
The authenticity of host 'thewalter.net (94.75.203.101)' can't be established.
RSA key fingerprint is 96:27:0c:85:a8:29:cc:a2:15:75:eb:55:18:7a:2c:29.
Are you sure you want to continue connecting (yes/no)?
```

Another example

Positive trust assertion

Subject is a public key.



eg: Certificate Revocation List

Serial+Issuer
Subject

Distrusted
Level of Trust

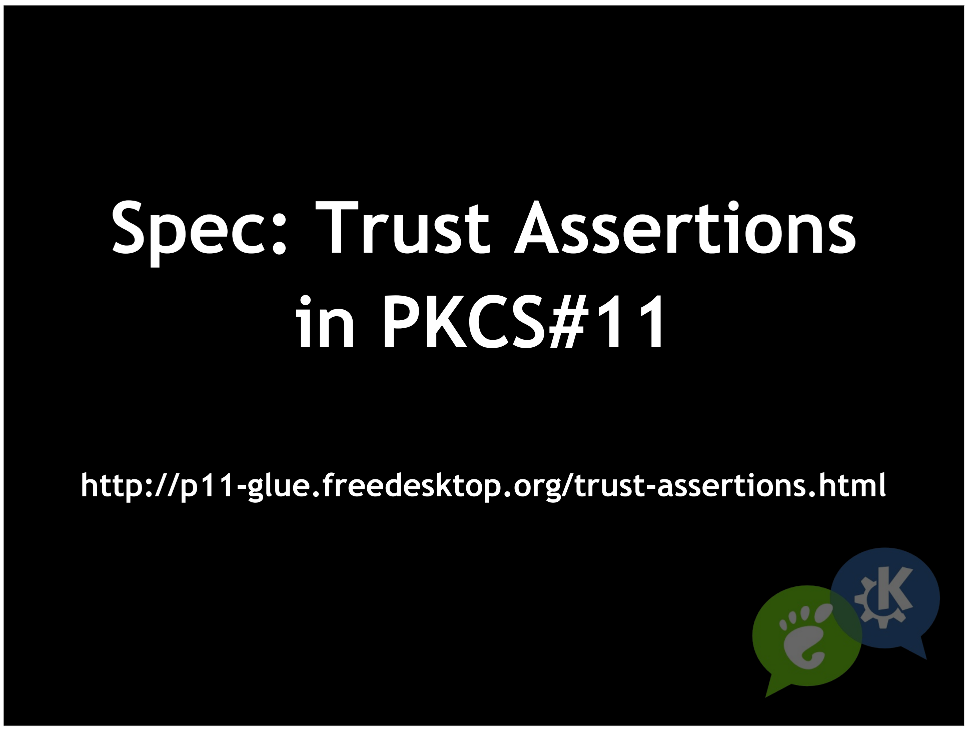
Any
Purpose

This time a negative trust assertion.

The subject is different.

The purpose is: any

Level of trust: distrusted.



Spec: Trust Assertions in PKCS#11

http://p11-glue.freedesktop.org/trust-assertions.html

We can use PKCS#11 to store trust assertions.

Trust assertions are stored in PKCS#11 objects, one object per trust assertion.

The various types of trust assertions in PKCS#11 is open ended and we can define new ones

Can be stored on a different token than the subject.

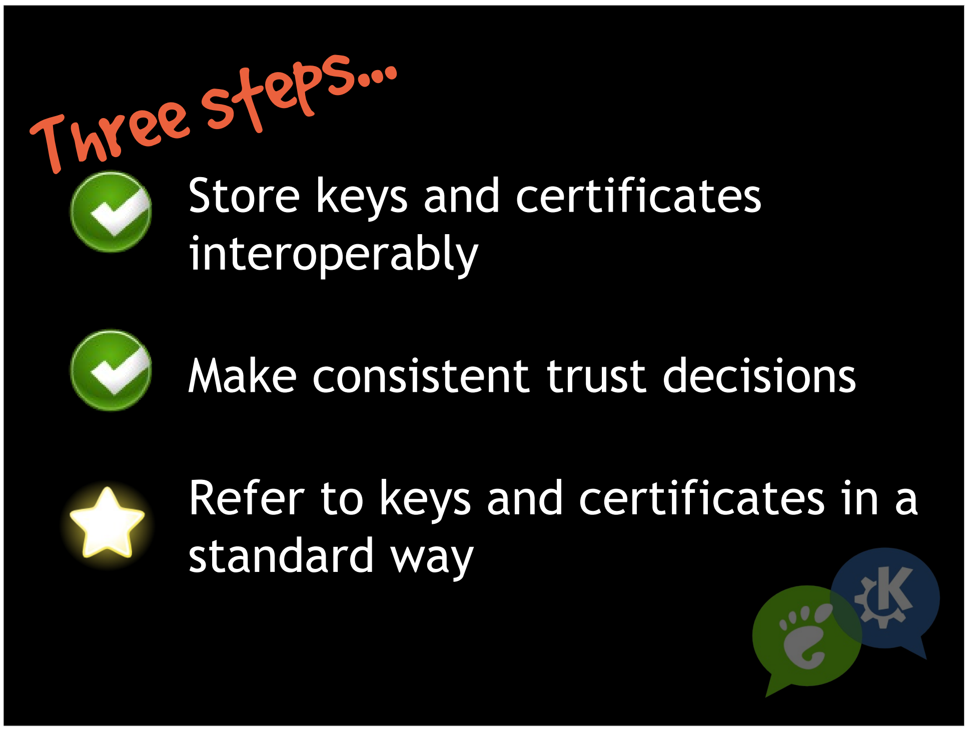
So this can work with any PKCS#11 provider, as long as there's one around that can store trust assertions.

By doing this we can:

1. Have all apps access the same trust assertions.

That doesn't mean they all make the same trust decisions. But they have the same data available to factor into their decisions.

2. Any source of trust assertions, can expose them as a PKCS#11 module, and apps and libraries can use them.



Three steps...

✓ Store keys and certificates interoperably

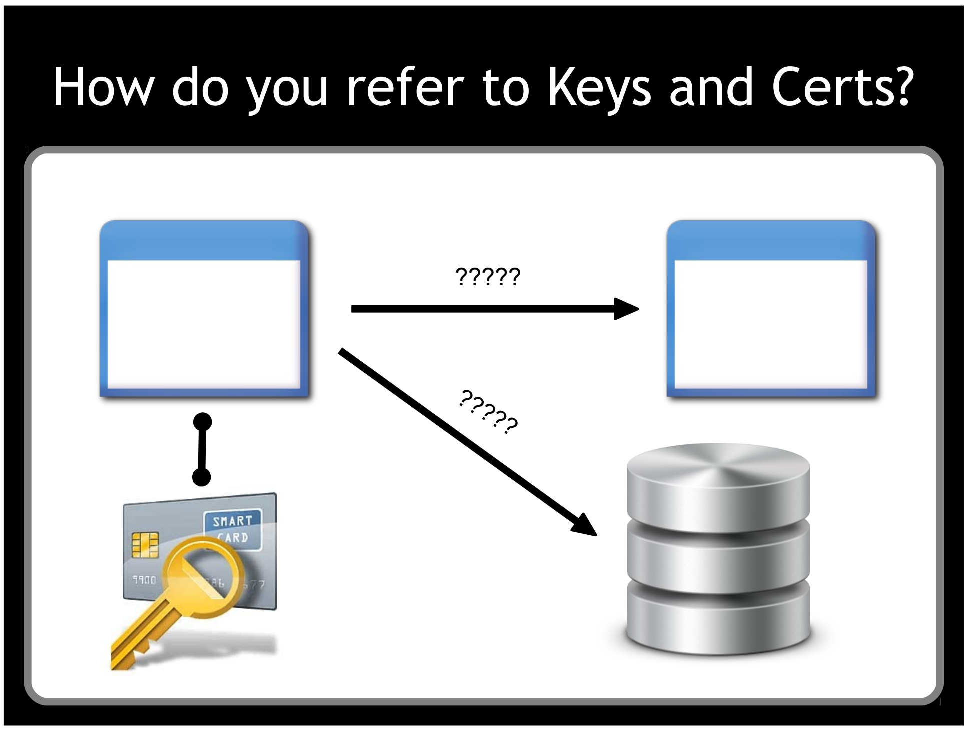
✓ Make consistent trust decisions

☆ Refer to keys and certificates in a standard way

Store keys and certificates in a way that is interoperable and multiple libraries and components can use them.

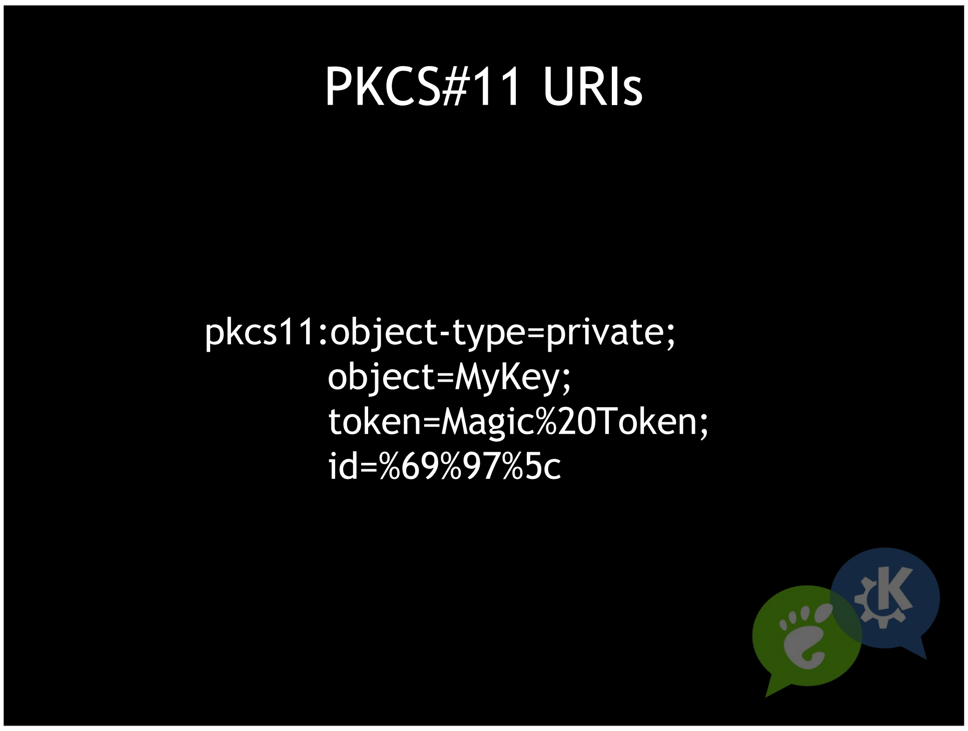
Make consistent trust decisions, so that the user doesn't have to try and figure out why one application 'works' while another doesn't, when doing the same thing.

Ability to for one application to refer to a key in a standard way, when it tells another library or application about it.



How do you refer to Keys and Certs?

Since we can't take keys out of a key store. How does one application communicate which certificate or key to use, to another application.



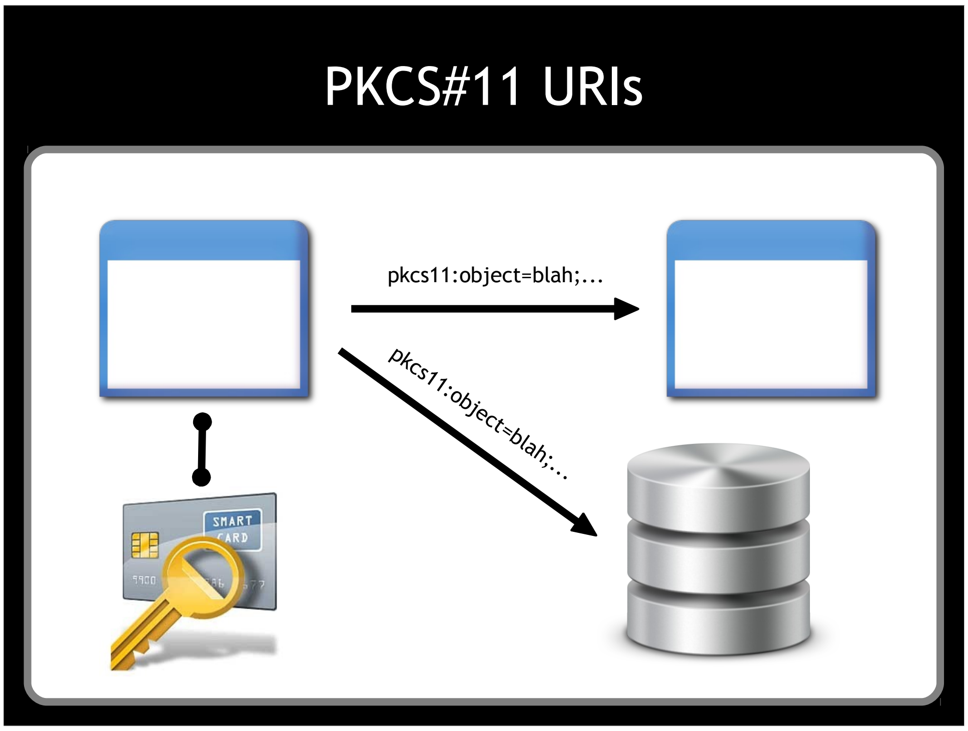
PKCS#11 URIs

```
pkcs11:object-type=private;
          object=MyKey;
          token=Magic%20Token;
          id=%69%97%5c
```

We use PKCS#11 URIs.

An RFC that we've been working on.

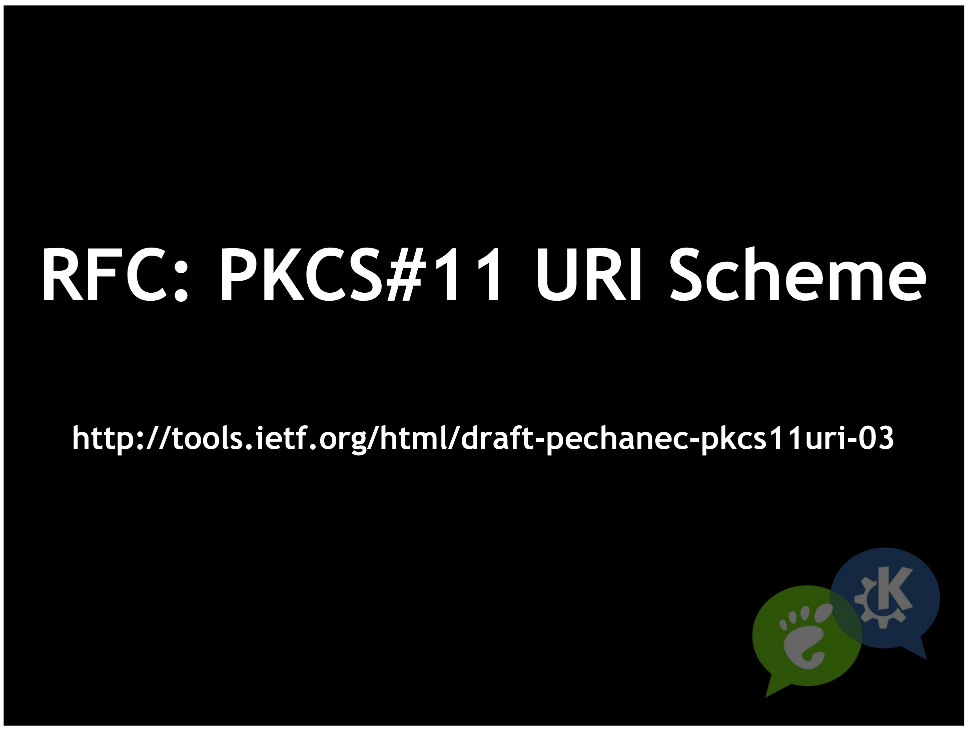
This string represents a key in a specific key store.



PKCS#11 URIs

We pass around these URIs instead of the keys themselves.

Since all apps can use the same PKCS#11 key stores, they can then use the same keys.

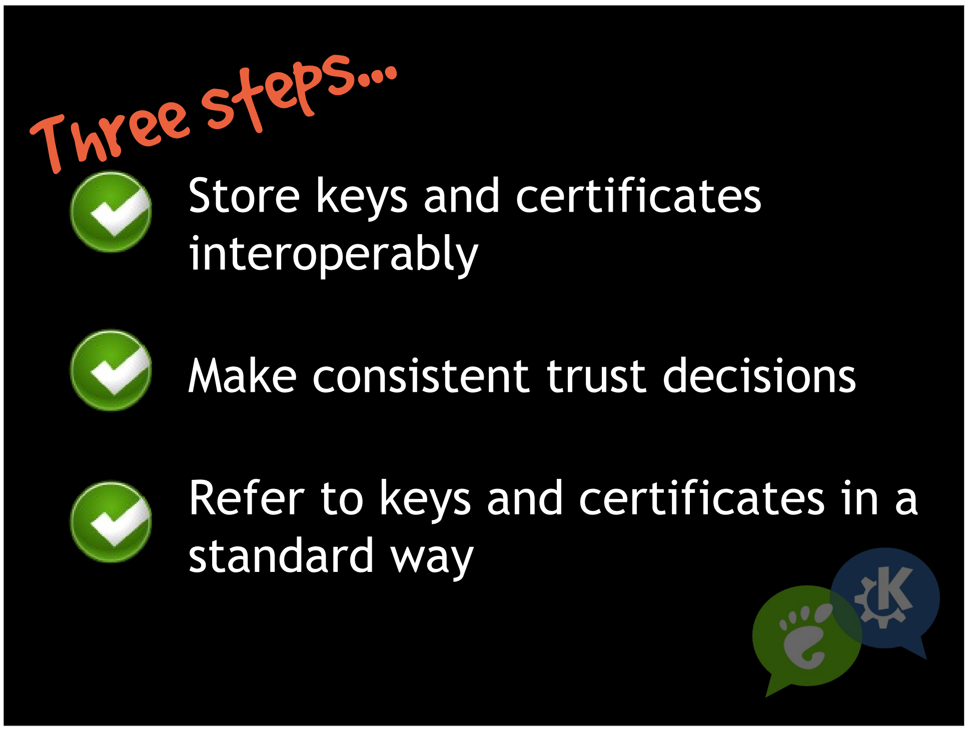


RFC: PKCS#11 URI Scheme

http://tools.ietf.org/html/draft-pechanec-pkcs11uri-03

RFC written by a couple of Sun (now oracle) guys, Jan and Daniel

A bunch of us contributed. Turning out quite nice.



Three steps...

- ✓ Store keys and certificates interoperably
- ✓ Make consistent trust decisions
- ✓ Refer to keys and certificates in a standard way

There we are.

Not actually exciting stuff, but I'm excited.

Once we get this boring stuff worked out, we can move to exciting stuff.

Attract users and developers to crypto, and get better solutions.

Not all of this is completely finalized and written in stone yet, so if you'd like to make your mark, and get your needs properly represented, join in the fun.

BTW, with all of this we're not throwing security to the wind. The architecture is designed so that the components can have access restrictions, and varying security or certification requirements.

Talk about glib support (half merged).

Gnutls support (all there in master, and gnutls 2.12 branches)

NSS support, works with proxies.

QT support, works with QCA (I haven't tested yet).

Any qvestions?

http://p11-glue.freedesktop.org

p11-glue@lists.freedesktop.org

- Icon: http://www.iconspedia.com/icon/smart-card-10843.html
- Thanks to everyone who contributed!

Collabora