Cryptographic Hardware Configuration and Features

Bruce Momjian

This presentation explains how cryptographic hardware uses private keys while preventing them from being viewed or copied.

 http://momjian.us/presentations

Last updated: September, 2018

Outline

1. What Problem Are We Trying To Solve? What Cryptography Can Accomplish

Authenticity Verify who is on the other end of the communication channel

Confidentiality Only the intended party can read the original messages

Integrity No other party can change or add messages

Non-repudiation Allow undeniable actions

The Problem With Passwords

- Passwords that are short enough to type and remember are easy enough to crack via brute force attacks
  - this is especially true if repeated password attempts are not blocked or if password attempts can be performed on a private copy of the data
- Uncrackable passwords are rarely rememberable or typeable

Updated password policies: https://www.theverge.com/2017/8/7/16107966/password-tips-bill-burr-regrets-advice-nits-cybersecurity

The Advantages of PKI

Before public key cryptography, the same secret key was required to encrypt and decrypt data.With public infrastructure (PKI), anyone to encrypt data and verify signatures using a public key. Only those with access to the private key can decrypt or sign documents.
Therefore, the security of the private key becomes critical. It is not typeable so it must be stored somewhere.

SSH PKI Keys

```
$ ls -la ~/.ssh
total 16
drwx------  2 postgres postgres 4096 Aug 14 10:25 .
drwxr-xr-x 22 postgres postgres 4096 Aug 14 10:25 ..
-rw-------  1 postgres postgres 1679 Aug 14 10:25 id_rsa
-rw-r--r--  1 postgres postgres  401 Aug 14 10:25 id_rsa.pub
```

Gpg PKI Keys

```
$ ls -la  ~/.gnupg/
total 36
drwx------  2 postgres postgres 4096 Aug 11 12:48 .
drwxr-xr-x 22 postgres postgres 4096 Aug 11 10:08 ..
-rw-------  1 postgres postgres 9188 Jul  5 15:23 gpg.conf
-rw-------  1 postgres postgres 1646 Jul  5 15:27 pubring.gpg
-rw-------  1 postgres postgres  600 Jul  5 15:27 random_seed
-rw-------  1 postgres postgres 1731 Jul  5 15:27 secring.gpg
-rw-------  1 postgres postgres 1280 Jul  5 15:27 trustdb.gpg
```

Postgres PKI Keys

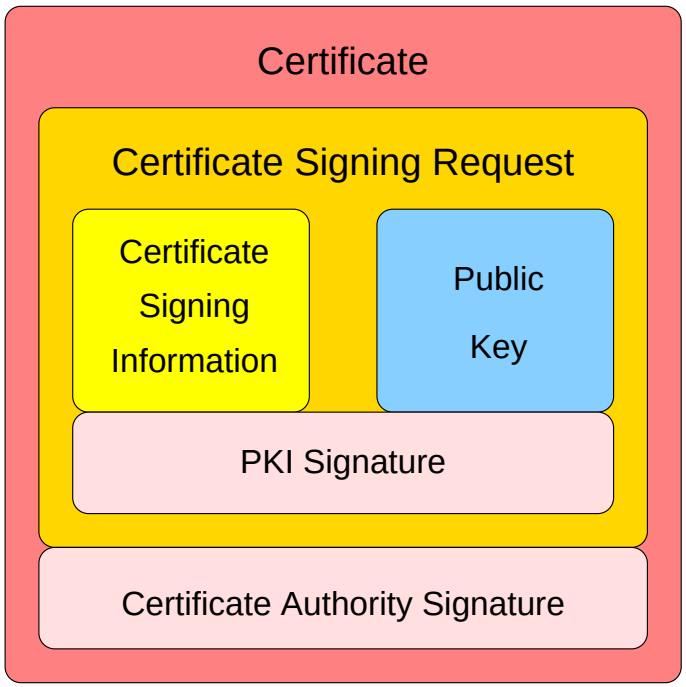
```
$ ls -la /u/pg/data/
total 128
drwx------ 19 postgres staff     4096 Aug 11 17:56 .
drwxr-sr-x  7 root     staff     4096 Aug 11 17:56 ..
…
-rw-r--r-- 1 root root 4087 Aug 11 12:50 /u/pg/data/server.crt
-rw------- 1 root root 1704 Aug 11 12:50 /u/pg/data/server.key
…
```

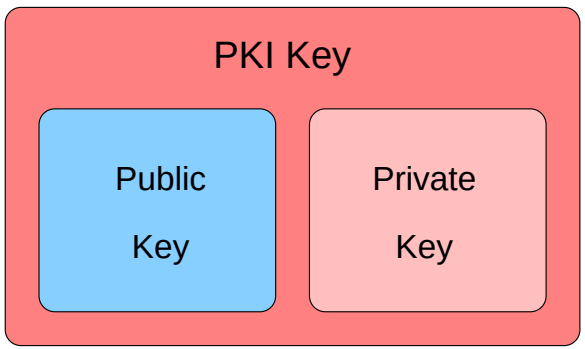
PKI Key Pair Internals

```
$ rsadump.sh -e id_rsa
key bits = 2047.2

# public
n (pq) ~= 1e616
e = 65537

# private
d (1/e mod lcm(p-1,q-1)) ~= 2e614
p ~= 1e308
q ~= 1e308

exp1 (dp, d mod (p-1)) ~= 9e307
exp2 (dq, d mod (q-1)) ~= 9e307
c (qinv, 1/q mod p) ~= 7e307
```

The final three fields are used to speed computations involving the private key. The private key, also called "key pair" or simply "key," also contains the public key. Script available at http://momjian.us/main/writings/pgsql/rsadump.sh, derived from http://www.vidarholen.net/contents/junk/files/decode_rsa.bash.

Certificate (with Public Key) and Public/Private Key Pair





Securing the Private Key

File system permissions protect the private key from ordinary users, but it does not protect it from unauthorized viewing, copying, and therefore use by:

- Operating system superusers
- Current and former employees
- Operating system exploits, trojans, and viruses

What Security Does PIV and OpenPGP with a Smart Card Provide?

- Use of a private key that cannot be viewed or copied by anyone, including the creator
- PIN access control with a limited number of failed attempts
- Removable private key storage

2. What Is a PIV Device?

PIV devices are US NIST-defined electronic containers with storage for owner:

- Identification
- Image
- Fingerprints
- Four PKI certificates and their keys

The standard defines how these items are accessed and manipulated:

- http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-73-4.pdf
- http://csrc.nist.gov/groups/SNS/piv/standards.html

PIV Interfaces

- USB (e.g., Yubikey)
- Card and card reader (e.g., CAC)
- Contactless

PIV Device: US Military Common Access Card (CAC)

https://en.wikipedia.org/wiki/Common_Access_Card
http://www.cac.mil/common-access-card/
https://www.army.mil/article/54310/dod-to-drop-social-security-numbers-from-id-cards

CAC (Card) Reader

http://www.af.mil/News/Article-Display/Article/127233/new-policy-protects-air-force-networks/

3. Features of the Yubikey 4 Nano

- The electronic properties of a PIV, including access control, but is not a card
- OpenPGP support
- USB interface, not a SIM chip
- No visually-identifying information
- Activity light and touch sensor
- The size of a fingernail

https://www.yubico.com/product/yubikey-4-series/

Yubikey 4 Security Features

- Creation of PKI keys on the device
- Import of externally-generated PKI keys
- Creation and import of certificates in PIV mode
- Stored keys can be used for decryption and signing, but never viewed or copied
- PIN controls use of the keys
- Three PIN failures blocks future use of the PIN until reset with a PUK (pin unblocking key)
- Three PUK failures requires a full device reset
- Certificates (with their public keys) can be viewed and exported anytime in PIV mode

https://developers.yubico.com/PIV/Introduction/Admin_access.html

Yubikey 4

https://en.wikipedia.org/wiki/YubiKey

Yubikey 4 Cryptographic Features

- Full PIV capabilities (already covered)
  - https://www.yubico.com/why-yubico/for-businesses/computer-login/yubikey-neo-and-piv/
  - https://developers.yubico.com/yubico-piv-tool/YubiKey_PIV_introduction.html
- OpenPGP support
- Brief-touch authentication slot
- Long-touch authentication slot

http://lorgor.blogspot.com/2016/09/yubikeys-demystified.html

Yubikey 4 Authentication-Only Options

- OATH
  - time-based one-time password algorithm (TOTP)
  - counter-based password algorithm (HOTP)
- Yubico one-time password (YOTP)
  - upload key to Yubico for later authentication
  - https://www.yubico.com/wp-content/uploads/2016/06/YubiKey_for_YubiCloud_ConfigGuide_en.pdf
- Unattended challenge-response
  - YOTP
  - HMAC-SHA1
  - other methods simulate keyboard input on touch
- Universal 2nd Factor (U2F), FIDO U2F
- Static password

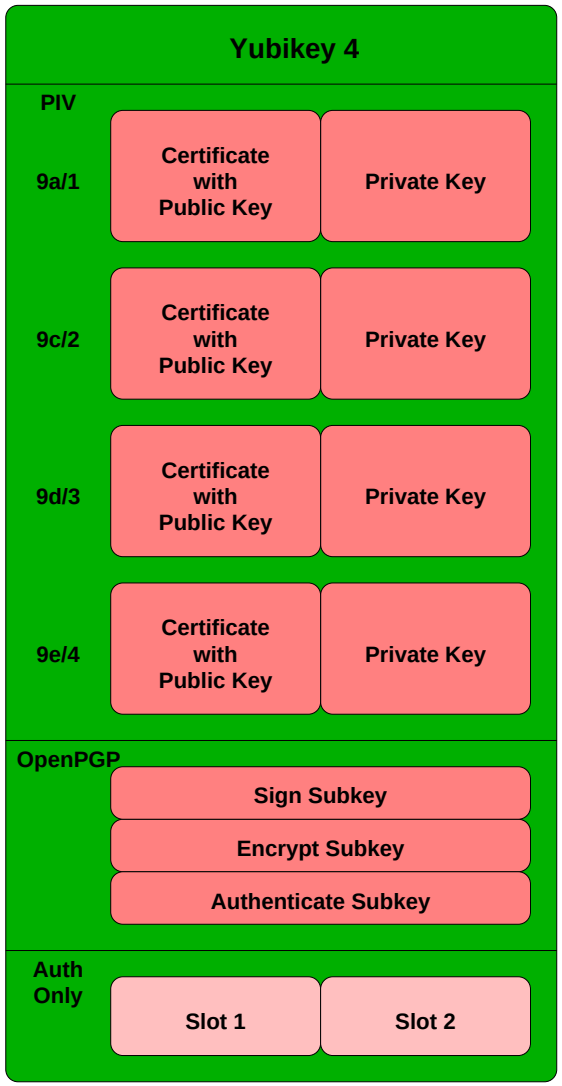
https://www.adfinis-sygroup.ch/blog/en/yubikey-and-twofactor-authentication/
https://www.adfinis-sygroup.ch/blog/en/yubikeys/

Yubikey 4 PIV Features

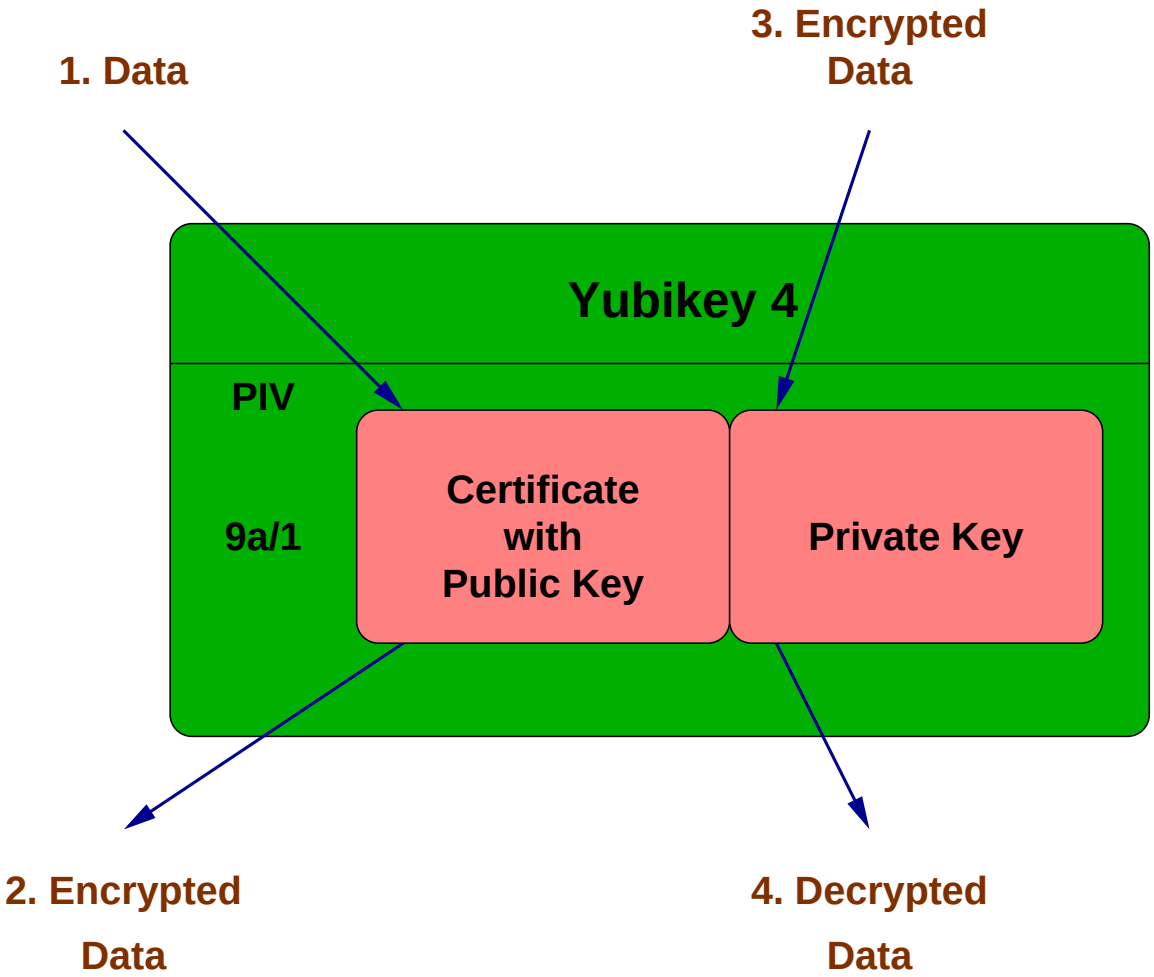
Description	*yubico-piv-tool*	*opensc/ openssl*	Private Key PIN Required	Decrypt	Sign
PIV Authentication	9a	1	✓	✓	✓
Triple-DES key (not used)	9b				
Digital Signature	9c	2	✓	✓	✓
Key Management	9d	3	✓	✓	
Card Authentication	9e	4			✓
Retired Key Management	82-8f, 90-95	n/a			
Attestation (prepopulated)	f9	n/a			

Openssl always requires a PIN; *pkcs15-crypt* honors the chart.
https://developers.yubico.com/PIV/Introduction/Certificate_slots.html

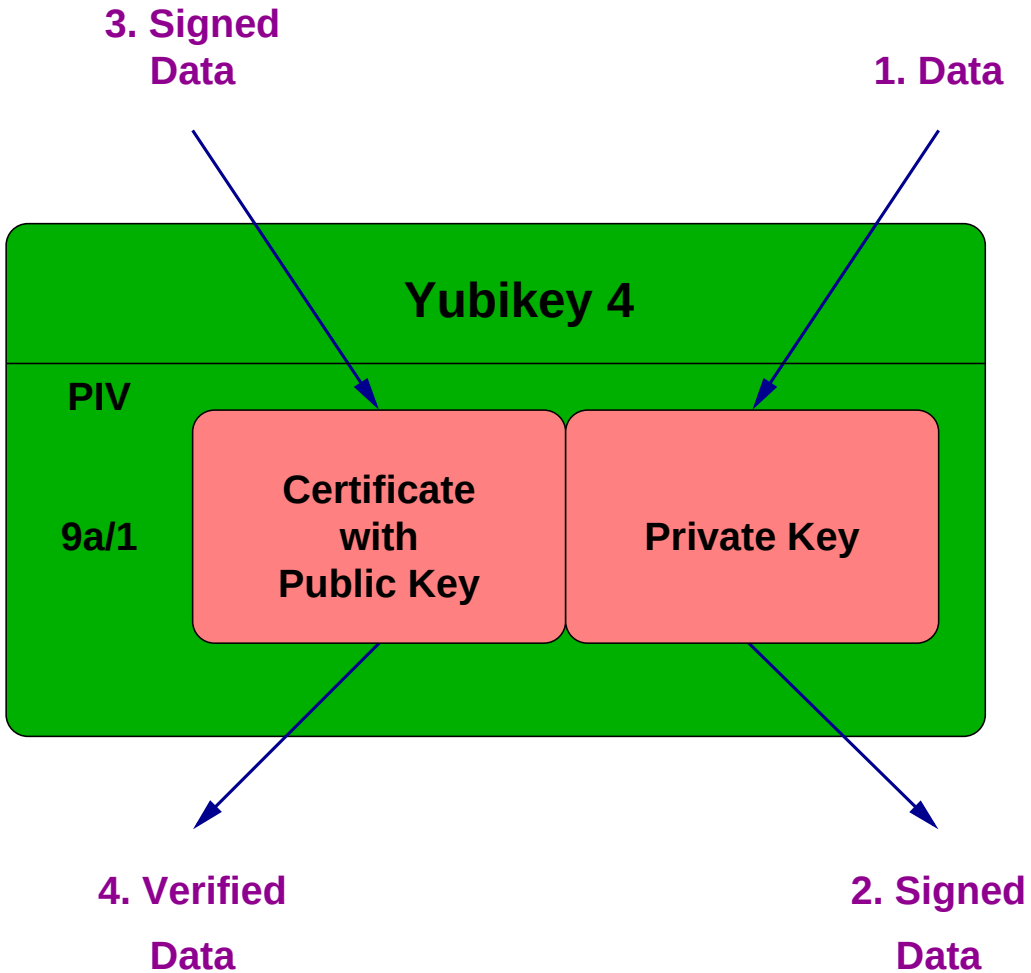
Yubikey 4 Illustrated



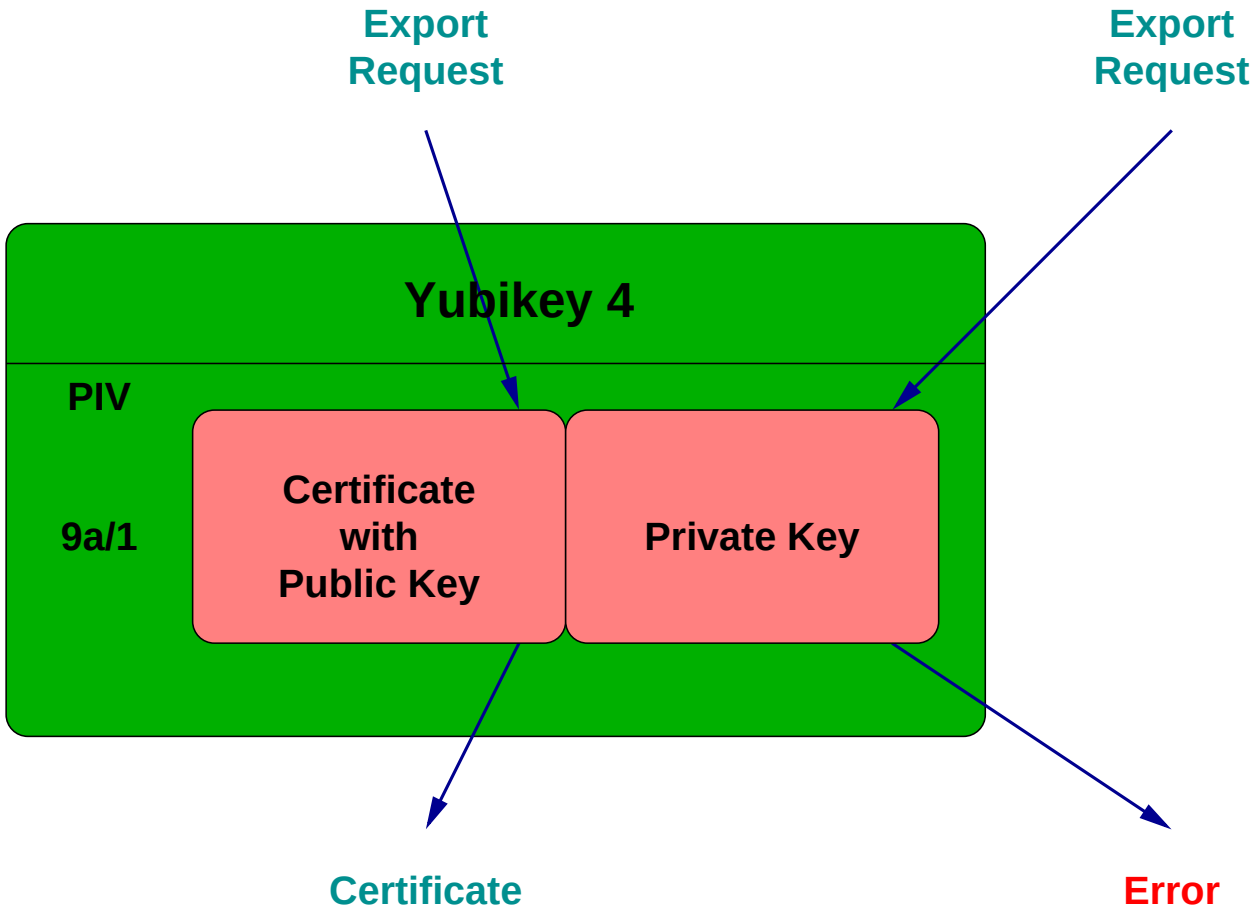
PIV Encryption



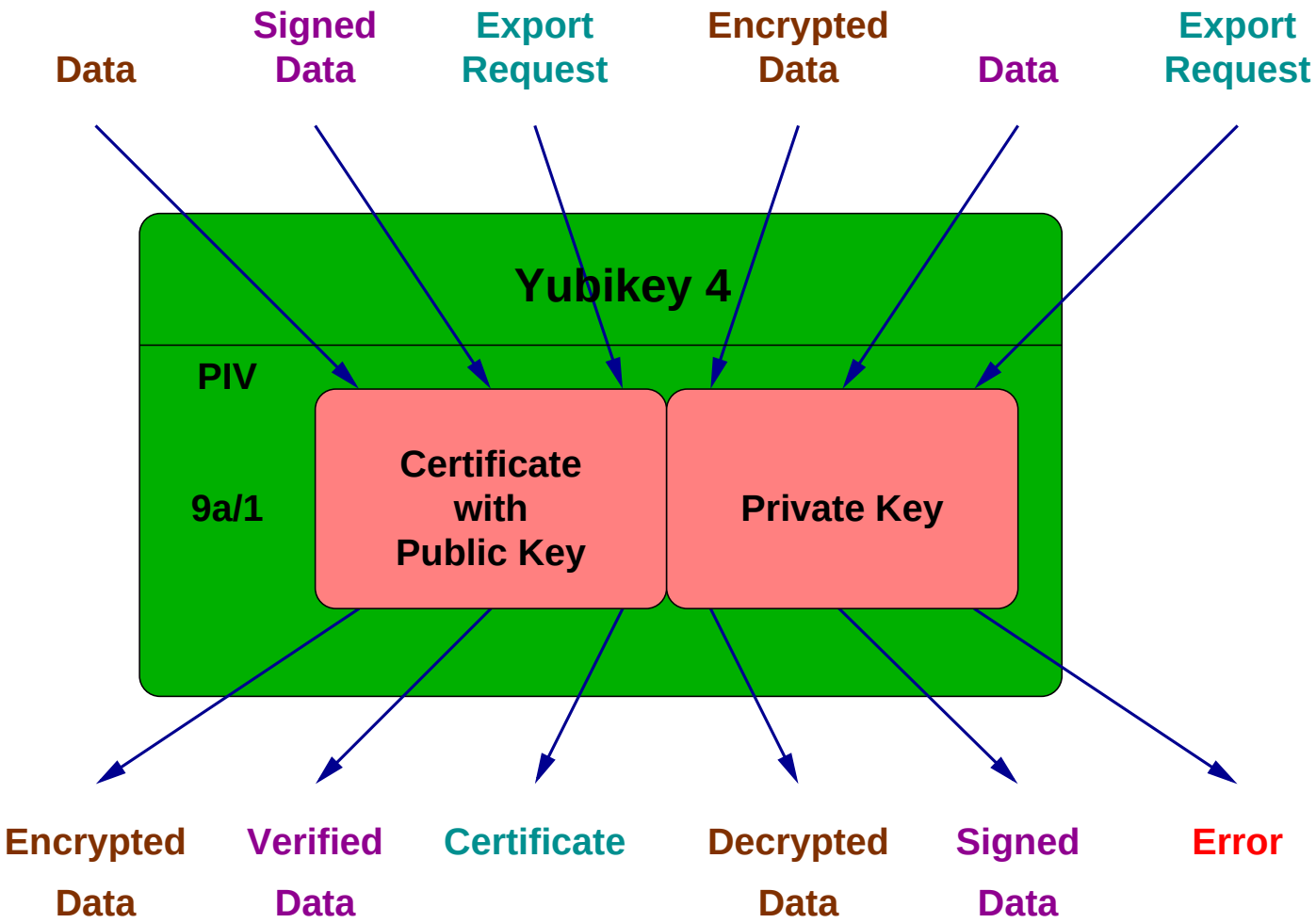
Piv Signing


3. Signed
Data
1. Data
Yubikey 4
PIV
9a/1
Certificate
with
Public Key
Private Key
4. Verified
Data
2. Signed
Data

PIV Exporting



All PIV Features



4. Software Configuration

1. Operating system software
2. USB configuration
3. *pcscd*
4. *openssl*
5. Yubikey configuration (not PIV configuration)

Configuration done using Debian 8 (Jessie) and Ubuntu 16.04 (Xenial Xerus).

Operating System Software

```
# Debian 8 (Jessie) needs the addition of '-t jessie-backports'
# Older operating systems use ppa:yubico/stable.
$ sudo apt-get install yubikey-personalization yubikey-personalization-gui

# also installs libccid, libykpiv1, and pcscd packages
# Debian 8 (Jessie) needs the addition of '-t jessie-backports'
$ sudo apt-get install yubico-piv-tool

# needed for PIV
# installs pkcs11-tool, pkcs15-tool, pkcs15-crypt
# also installs engine_pkcs11 package
$ sudo apt-get install opensc
$ sudo apt-get install libengine-pkcs11-openssl
```

Update USB and CCID Configuration Files

```
# Update CCID for device detection
# modifies /etc/libccid_Info.plist and /etc/udev/rules.d/70-u2f.rules
# download script from https://raw.githubusercontent.com/Yubico/
yubikey-neo-manager/master/resources/linux-fix-ccid-udev
$ chmod +x ./linux-fix-ccid-udev
$ sudo ./linux-fix-ccid-ude
Updating /etc/libccid_Info.plist...
Restarting PCSCD...
Adding Udev rule for U2F...
Done!
```

Access Control

```
$ sudo adduser yubiuser

# modify /lib/udev/rules.d/69-yubikey.rules to set USB character device
# permissions (in red)
$ sudo vi /lib/udev/rules.d/69-yubikey.rules

ATTRS{idVendor}=="1050", \
ATTRS{idProduct}=="0010|0110|0111|0114|0116|0401|0403|0405|0407|0410", \
ENV{ID_SECURITY_TOKEN}="1", OWNER="yubiuser", GROUP="yubiuser", MODE="0660"
```

http://weininger.net/how-to-write-udev-rules-for-usb-devices.html
http://www.reactivated.net/writing_udev_rules.html

Checking Character Device Permissions

```
$ lsusb
…
Bus 002 Device 002: ID 1050:0407 Yubico.com

$ sudo udevadm test $(udevadm info -q path -n /dev/bus/usb/002/002)
$ ls -l /dev/bus/usb/002/002
crw-rw---- 1 yubiuser yubiuser  189, 132 Aug 28 16:44 /dev/bus/usb/002/002
```

pcscd Configuration

By default, *pcscd* creates an unrestricted socket to interface to the card, so set the socket permissions:

```
# modify /lib/systemd/system/pcscd.socket to set socket permissions (in red)
$ sudo vi /lib/systemd/system/pcscd.socket


[Socket]
ListenStream=/var/run/pcscd/pcscd.comm
SocketUser=yubiuser
SocketGroup=yubiuser
SocketMode=0660
```

See *man systemd.socket*

Checking *pcscd* Socket Permissions

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart pcscd.socket
$ ls -l /var/run/pcscd/pcscd.comm
srw-rw---- 1 yubiuser yubiuser  0 Aug 29 08:03 /var/run/pcscd/pcscd.comm
```

The Importance of Access Control

While it is true that private key information cannot be extracted from the Yubikey device, and requires a PIN for its use, unauthorized access can:

- Prevent use and force a device reset by exceeding the PIN invalid entry limit
- Reset the device and reprogram it with replacement keys that might not be immediately detected

http://musclecard.996296.n3.nabble.com/Restricting-reader-card-access-by-user-account-td431.html#a457
https://access.redhat.com/blogs/766093/posts/1976313

openssl Configuration

```
$ sudo vi /etc/ssl/openssl.cnf

# add before any sections are defined file
        openssl_conf = openssl_init

# and add these sections at the bottom
        [openssl_init]
        engines = engine_section
        [engine_section]
        pkcs11 = pkcs11_section
        [pkcs11_section]
        # https://github.com/openssl/openssl/blob/master/README.ENGINE
        engine_id = pkcs11
        # same as SO_PATH
        dynamic_path = /usr/lib/engines/engine_pkcs11.so
        MODULE_PATH = opensc-pkcs11.so
        init = 0
$ openssl engine pkcs11 -t
(pkcs11) pkcs11 engine
     [ available ]
```

https://github.com/OpenSC/libp11
https://stackoverflow.com/questions/15052171/pkcs11-engine-does-not-work-in-openssl-on-centos-6

Yubikey Configuration

```
# Enable OTP/UTF/CCID mode
# Removing OTP mode also disables communication between ykpersonalize and
# the YubiKey.
$ ykpersonalize -y -m 6


# Erase the two authorization-only slots
$ ykpersonalize -y -1 -z
$ ykpersonalize -y -2 -z


$ ykinfo -a
serial: 6251544
serial_hex: 5f6418
serial_modhex: gvhfbj
version: 4.3.5
touch_level: 512
programming_sequence: 0
slot1_status: 0
slot2_status: 0
vendor_id: 1050
product_id: 407
```

https://developers.yubico.com/yubikey-personalization/Manuals/ykpersonalize.1.html

5. PIV Configuration: Check *opensc*

```
$ opensc-tool --list-drivers | grep '\<piv\>'
  piv               PIV-II  for multiple cards

$ opensc-tool --info
OpenSC 0.14.0 [gcc  4.9.2]
Enabled features: zlib readline openssl pcsc(libpcsclite.so.1)

$ sudo opensc-tool --name
Using reader with a card: Yubico Yubikey 4 OTP+U2F+CCID 00 00
PIV-II card
```

https://github.com/OpenSC/OpenSC/wiki/Quick-Start-with-OpenSC
https://stackoverflow.com/questions/19456555/openssl-engine-pkcs11-libp11-opensc

Reset the PIV Mode

```
#!/bin/bash

# must be run by a user in the yubiuser group

# Attempt to use an invalid PIN multiple times to block it #
yubico-piv-tool -a verify-pin -P 000000
yubico-piv-tool -a verify-pin -P 000000
yubico-piv-tool -a verify-pin -P 000000
yubico-piv-tool -a verify-pin -P 000000

# Attempt to change PUK using an invalid PUK multiple times to block it #
yubico-piv-tool -a change-puk -P 000000 -N 000001
yubico-piv-tool -a change-puk -P 000000 -N 000001
yubico-piv-tool -a change-puk -P 000000 -N 000001
yubico-piv-tool -a change-puk -P 000000 -N 000001

# Once PIN and PUK are both blocked, you can reset the YubiKey.
yubico-piv-tool -a reset
```

https://developers.yubico.com/yubico-piv-tool/YubiKey_PIV_introduction.html
https://www.yubico.com/wp-content/uploads/2016/05/Yubico_PIV_Tool_Command_Line_Guide_en.pdf

Set configuration key, PUK, and PIN

```
#!/bin/bash

# don't write Bash shell history if run interactively
unset HISTFILE

cd "$HOME" || exit 1
umask 0077

mkdir .yubikey 2> /dev/null
rm -f .yubikey/pin.*
```

Set configuration key, PUK, and PIN

```
CONFIG_KEY=''$(dd if=/dev/random bs=1 count=24 2>/dev/null |
              hexdump -v -e '/1 "%02X"')''
yubico-piv-tool -aset-mgm-key -n $CONFIG_KEY
echo "$CONFIG_KEY" | tee .yubikey/piv.config.key

yubico-piv-tool -k $CONFIG_KEY -a verify -P 123456 -a pin-retries \
                --pin-retries=3 --puk-retries=3

PUK=''$(dd if=/dev/random bs=1 count=6 2>/dev/null |
       hexdump -v -e '/1 "%u"'|cut -c1-8)''
yubico-piv-tool -achange-puk -P 12345678 -N $PUK
echo "$PUK" | tee .yubikey/piv.puk

PIN=''$(dd if=/dev/random bs=1 count=6 2>/dev/null |
           hexdump -v -e '/1 "%u"'|cut -c1-6)''
yubico-piv-tool -achange-pin -P 123456 -N $PIN
echo "$PIN" | tee .yubikey/piv.pin
```

Populate the Four PIV Slots

```
#! /bin/bash
unset HISTFILE

$ cd "$HOME" || exit 1

CONFIG_KEY="$(cat .yubikey/piv.config.key)"
PIN="$(cat .yubikey/piv.pin)"
echo "$CONFIG_KEY"
echo "$PIN"

cd .yubikey || exit 1

# https://developers.yubico.com/yubico-piv-tool/
# https://dennis.silvrback.com/openssl-ca-with-yubikey-neo
```

Populate the Four PIV Slots

```
for SLOT in 9a 9c 9d 9e
do      if [ "$SLOT" = "9a" ]
        # Generate the key in the Yubikey
        # 9a is the authentication slot, so we can regenerate it if
        # we lose it.
        then    yubico-piv-tool --key "$CONFIG_KEY" -a generate -s "$SLOT" -o pub
                # The public key is just dumped in the slot so we have
                # to create a real certificate.
                yubico-piv-tool --pin-policy=once --pin $PIN -i pub \
                        -a verify -a selfsign-certificate \
                        -s "$SLOT" -S "/CN=yubikey-$SLOT" -o cert
        # Generate the key on the server
        # We want to archive the values for these slots, so we generate the key
        # externally so we can save it before putting it on the card.
        else    openssl req -x509 -nodes -newkey rsa:2048 \
                -subj "/CN=yubikey-$SLOT" -days 365 -keyout key -out cert
                yubico-piv-tool --pin-policy=once --key "$CONFIG_KEY" \
                                -i key -a import-key -s "$SLOT"
        fi


        yubico-piv-tool --key "$CONFIG_KEY" --pin-policy=once -i cert \
                        -a import-certificate -s "$SLOT"
        rm -f key pub cert
done
```

Key Generation Options

- Inside the Yubikey 4 (no private key backup possible)
- *openssl {req | genpkey}*
- *ssh-keygen (openssh)*
- *gpg2 –gen-key*

We used the first two in the previous slide.

Check the PIV Device

```
$ yubico-piv-tool -a status
CHUID:  No data available
CCC:    No data available
PIN tries left: 3


if dpkg -l opensc-pkcs11 > /dev/null 2>&1
> then    PACKAGE="opensc-pkcs11" # newer packaging
> else    PACKAGE="opensc"
> fi
$ export OPENSC="$(dpkg -L "$PACKAGE" | grep '/opensc-pkcs11.so$' | head -1)"


$ pkcs11-tool --module "$OPENSC" --list-slots
Available slots:
Slot 0 (0xffffffffffffffff): Virtual hotplug slot
  (empty)
Slot 1 (0x1): Yubico Yubikey 4 OTP+U2F+CCID 00 00
  token label        : PIV_II (PIV Card Holder pin)
  token manufacturer : piv_II
  token model        : PKCS#15 emulated
  token flags        : rng, login required, PIN initialized, token initialized
  hardware version   : 0.0
  firmware version   : 0.0
  serial num         : 00000000
```

List PIV Features

```
$ pkcs11-tool --module "$OPENSC" --list-mechanisms
Using slot 1 with a present token (0x1)
Supported mechanisms:
  SHA-1, digest
  SHA256, digest
  SHA384, digest
  SHA512, digest
  MD5, digest
  RIPEMD160, digest
  GOSTR3411, digest
  ECDSA, keySize={256,384}, hw, sign, other flags=0x1800000
  ECDSA-SHA1, keySize={256,384}, hw, sign, other flags=0x1800000
  ECDH1-COFACTOR-DERIVE, keySize={256,384}, hw, derive, other flags=0x1800000
  ECDH1-DERIVE, keySize={256,384}, hw, derive, other flags=0x1800000
  RSA-X-509, keySize={1024,3072}, hw, decrypt, sign, verify
  RSA-PKCS, keySize={1024,3072}, hw, decrypt, sign, verify
  SHA1-RSA-PKCS, keySize={1024,3072}, sign, verify
  SHA256-RSA-PKCS, keySize={1024,3072}, sign, verify
  MD5-RSA-PKCS, keySize={1024,3072}, sign, verify
  RIPEMD160-RSA-PKCS, keySize={1024,3072}, sign, verify
```

List PIV Objects

```
$ pkcs11-tool --module "$OPENSC" --list-objects|less
Data object 11293952
  label:          'Card Capability Container'
  application:    'Card Capability Container'
  app_id:         2.16.840.1.101.3.7.1.219.0
  flags:          <empty>

Data object 11295440
  label:          'Card Holder Unique Identifier'
  application:    'Card Holder Unique Identifier'
  app_id:         2.16.840.1.101.3.7.2.48.0
  flags:          <empty>

Data object 11296192
  label:          'Unsigned Card Holder Unique Identifier'
  application:    'Unsigned Card Holder Unique Identifier'
  app_id:         2.16.840.1.101.3.7.2.48.2
  flags:          <empty>
```

List PIV Certificates

```
$ pkcs15-tool --list-certificates
X.509 Certificate [Certificate for PIV Authentication]
        Object Flags   : [0x0]
        Authority      : no
        Path           :
        ID             : 01
        Encoded serial : 02 09 00CF66D8199E854D7A
X.509 Certificate [Certificate for Digital Signature]
        Object Flags   : [0x0]
        Authority      : no
        Path           :
        ID             : 02
        Encoded serial : 02 09 00FCDD2A4D4B080F64
X.509 Certificate [Certificate for Key Management]
        Object Flags   : [0x0]
        Authority      : no
        Path           :
        ID             : 03
        Encoded serial : 02 09 00AAB47AC5A07BD312
X.509 Certificate [Certificate for Card Authentication]
        Object Flags   : [0x0]
        Authority      : no
        Path           :
        ID             : 04
        Encoded serial : 02 09 0083DEF6124008A6BC
```

List PIV Public Keys

```
$ pkcs15-tool --list-public-keys
Using reader with a card: Yubico Yubikey 4 OTP+U2F+CCID 00 00
Public RSA Key [PIV AUTH pubkey]
        Object Flags   : [0x0]
        Usage          : [0xD1], encrypt, wrap, verify, verifyRecover
        Access Flags   : [0x2], extract
        ModLength      : 2048
        Key ref        : 154 (0x9A)
        Native         : yes
        Auth ID        : 01
        ID             : 01
        DirectValue    : <absent>

Public RSA Key [SIGN pubkey]
        Object Flags   : [0x0]
        Usage          : [0x2C1], encrypt, verify, verifyRecover, nonRepudiation
        Access Flags   : [0x2], extract
        ModLength      : 2048
        Key ref        : 156 (0x9C)
        Native         : yes
        Auth ID        : 01
        ID             : 02
        DirectValue    : <absent>
```

List PIV Public Keys

```
Public RSA Key [KEY MAN pubkey]
        Object Flags    : [0x0]
        Usage           : [0x11], encrypt, wrap
        Access Flags    : [0x2], extract
        ModLength       : 2048
        Key ref         : 157 (0x9D)
        Native          : yes
        Auth ID         : 01
        ID              : 03
        DirectValue     : <absent>

Public RSA Key [CARD AUTH pubkey]
        Object Flags    : [0x0]
        Usage           : [0xC0], verify, verifyRecover
        Access Flags    : [0x2], extract
        ModLength       : 2048
        Key ref         : 158 (0x9E)
        Native          : yes
        Auth ID         : 00
        ID              : 04
        DirectValue     : <absent>
```

List PIV Private Keys

```
$ pkcs15-tool --list-keys
Using reader with a card: Yubico Yubikey 4 OTP+U2F+CCID 00 00
Private RSA Key [PIV AUTH key]
        Object Flags   : [0x1], private
        Usage          : [0x2E], decrypt, sign, signRecover, unwrap
        Access Flags   : [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength      : 2048
        Key ref        : 154 (0x9A)
        Native         : yes
        Auth ID        : 01
        ID             : 01

Private RSA Key [SIGN key]
        Object Flags   : [0x1], private
        Usage          : [0x20E], decrypt, sign, signRecover, nonRepudiation
        Access Flags   : [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength      : 2048
        Key ref        : 156 (0x9C)
        Native         : yes
        Auth ID        : 01
        ID             : 02
```

List PIV Private Keys

```
Private RSA Key [KEY MAN key]
        Object Flags    : [0x1], private
        Usage           : [0x22], decrypt, unwrap
        Access Flags    : [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength       : 2048
        Key ref         : 157 (0x9D)
        Native          : yes
        Auth ID         : 01
        ID              : 03

Private RSA Key [CARD AUTH key]
        Object Flags    : [0x0]
        Usage           : [0xC], sign, signRecover
        Access Flags    : [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength       : 2048
        Key ref         : 158 (0x9E)
        Native          : yes
        ID              : 04
```

Test Piv

```
$ pkcs11-tool --module "$OPENSC" --pin "$(cat "$HOME"/.yubikey/piv.pin)" --login --
Using slot 1 with a present token (0x1)
C_SeedRandom() and C_GenerateRandom():
  seeding (C_SeedRandom) not supported
  seems to be OK
Digests:
  all 4 digest functions seem to work
  MD5: OK
  SHA-1: OK
  RIPEMD160: OK
Signatures (currently only RSA signatures)
  testing key 0 (PIV AUTH key)
  all 4 signature functions seem to work
  testing signature mechanisms:
    RSA-X-509: OK
    RSA-PKCS: OK
    SHA1-RSA-PKCS: OK
    MD5-RSA-PKCS: OK
    RIPEMD160-RSA-PKCS: OK
    SHA256-RSA-PKCS: OK
  testing key 1 (2048 bits, label=SIGN key) with 1 signature mechanism
    MD5-RSA-PKCS: OK
  testing key 2 (2048 bits, label=KEY MAN key) with 1 signature mechanism -- can't
  testing key 3 (2048 bits, label=CARD AUTH key) with 1 signature mechanism
```

Test PIV

```
# Debian 8 (Jessie) fails some of these tests
Verify (currently only for RSA):
  testing key 0 (PIV AUTH key)
    RSA-X-509: OK
    RSA-PKCS: OK
    SHA1-RSA-PKCS: OK
    MD5-RSA-PKCS: OK
    RIPEMD160-RSA-PKCS: OK
  testing key 1 (SIGN key) with 1 mechanism
    RSA-X-509: OK
  testing key 2 (KEY MAN key) with 1 mechanism
 -- can't be used to sign/verify, skipping
  testing key 3 (CARD AUTH key) with 1 mechanism
    RSA-X-509: OK
Unwrap: not implemented
Decryption (RSA)
  testing key 0 (PIV AUTH key)
    RSA-X-509: OK
    RSA-PKCS: OK
  testing key 1 (SIGN key)
    RSA-X-509: OK
    RSA-PKCS: OK
  testing key 2 (KEY MAN key)
    RSA-X-509: OK
    RSA-PKCS: OK
```

6. *Openssl* with PIV

```
$ unset HISTFILE

$ PIN="$(cat "$HOME"/.yubikey/piv.pin)"

# 'pkeyutl' and 'dgst' generate "unimplemented" errors in
# ''OpenSSL 1.0.1f 6 Jan 2014'' and ''OpenSSL 1.0.1t  3 May 2016''
# 'rsautl' can only process a limited length of data
# also 'slot_1-id_3' or ''label_KEY MAN key'' for -inkey
# see 'man openssl' for --passin arguments
$ echo test |
> openssl rsautl -engine pkcs11 -keyform engine -encrypt -inkey 1:3 \
>                              -passin file:"$HOME"/.yubikey/piv.pin -out /tmp/crypt

$ openssl rsautl -engine pkcs11 -keyform engine -decrypt -inkey 1:3 \
>                 -passin file:"$HOME"/.yubikey/piv.pin -in /tmp/crypt
test
```

Test With *openssl* and *pkcs15-crypt*

```
$ echo test |
> openssl rsautl -engine pkcs11 -keyform engine -encrypt -inkey 1:3 \
>                -passin file:"$HOME"/.yubikey/piv.pin -out /tmp/crypt

# use stdout so \n is properly output
$ pkcs15-crypt --decipher -i /tmp/crypt -o /dev/stdout --pkcs1 -p $PIN --key 3
test
```

Test by Extracting the Certificate

```
# Yubikeys store the certificate (with public key) and the private key,
# not the public key alone.
$ pkcs15-tool --read-certificate 3 > /tmp/cert

$ echo test |
> openssl pkeyutl -encrypt -certin -inkey /tmp/cert -out /tmp/crypt

$ pkcs15-crypt --decipher -i /tmp/crypt -o /dev/stdout --pkcs1 -p $PIN --key 3
test
```

Test Using a Hash

```
$ echo test |
> openssl dgst -sha256 -binary > /tmp/hash
$ xxd -p -c 999 /tmp/hash
f2ca1bb6c7e907d06dafe4687e579fce76b37e4e93b7605022da52e6ccc26fd2

$ openssl rsautl -engine pkcs11 -keyform engine -encrypt -inkey 1:3 \
>                -passin file:"$HOME"/.yubikey/piv.pin -in /tmp/hash \
>                -out /tmp/crypt
$ openssl rsautl -engine pkcs11 -keyform engine -decrypt -inkey 1:3 \
>                -passin file:"$HOME"/.yubikey/piv.pin -in /tmp/crypt |
>                xxd -p -c 999
f2ca1bb6c7e907d06dafe4687e579fce76b37e4e93b7605022da52e6ccc26fd2
```

Certificates can be signed, https://developers.yubico.com/PIV/Guides/Certificate_authority.html

Test Signing

```
# switch to using the signing key (2, ''label_SIGN key'')
$ echo test |
> openssl rsautl -engine pkcs11 -keyform engine -sign -inkey 1:2 \
>                -passin file:"$HOME"/.yubikey/piv.pin -out /tmp/sign

$ openssl rsautl -engine pkcs11 -keyform engine -verify -inkey 1:2 \
                 -passin file:"$HOME"/.yubikey/piv.pin -in /tmp/sign
test
```

Software and Standards Summary

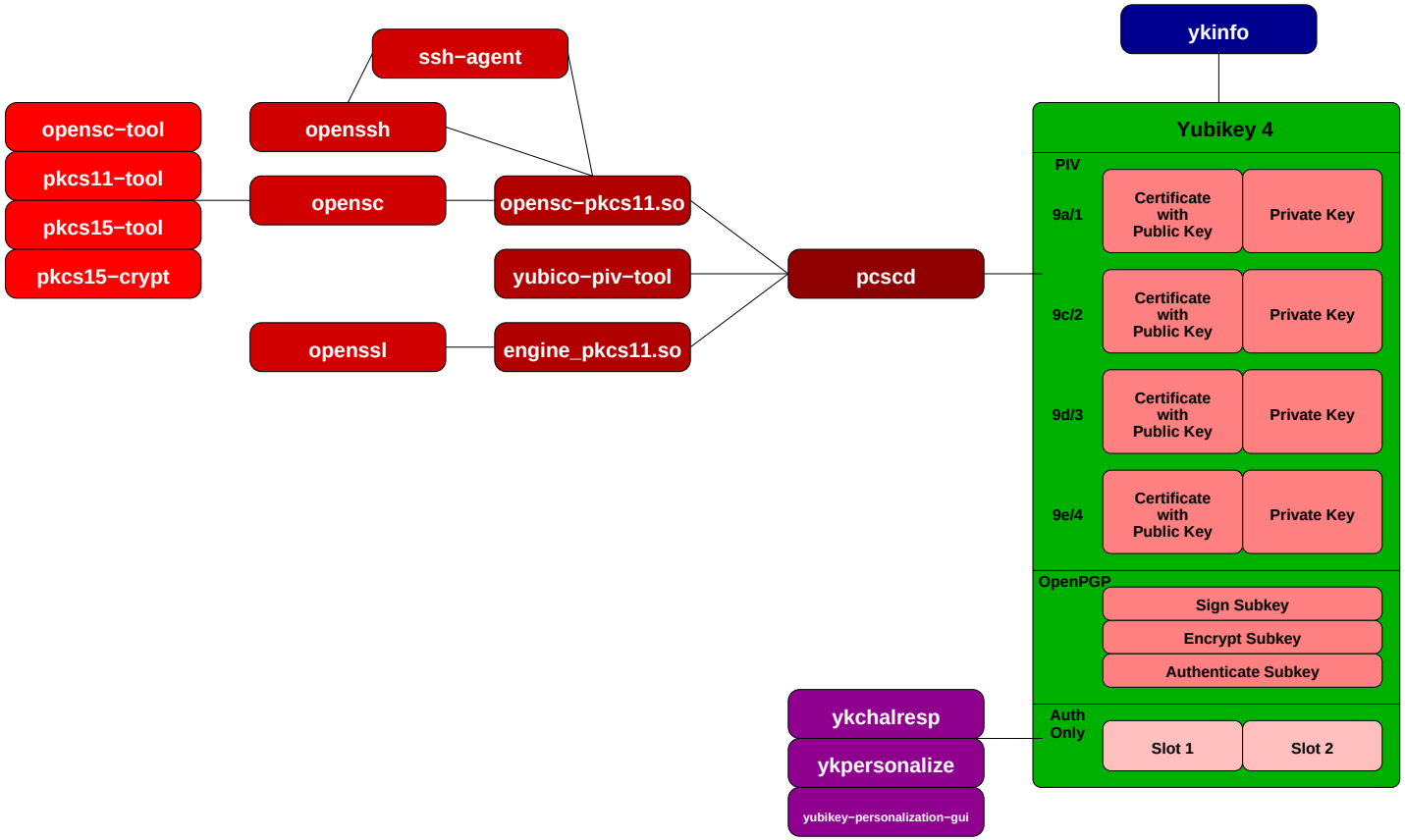
CCID	a USB smart card interface standard
engine_pkcs11.so	library to interface OpenSSL to PKCS#11 smart cards
OpenSC	provides a standard interface to smart cards like PKCS#11
opensc-pkcs11.so	library to interface OpenSC to PKCS#11 smart cards
OpenSSL	TLS/SSL library
pcscd	PC/SC smart card daemon
PIV	smart card standard from us NIST
PKCS#11	software interface to cryptographic tokens
PKCS#15	token information format

Tool Summary

opensc-tool	smart card operations
openssl	TLS/SSL operations
pkcs11-tool	read and manipulate the PIV container
pkcs15-tool	read and manipulate PIV objects
pkcs15-crypt	control PIV decryption and signing
ykchalresp	perform YubiKey challenge-response operations
ykinfo	report Yubikey settings
ykpersonalize	Yubikey general (non-PIV) operations
yubico-piv-tool	Yubikey PIV operations
yubikey-personalization-gui	GUI for Yubikey operations

https://blog-ftweedal.rhcloud.com/2016/08/smart-card-login-with-yubikey-neo/

Software and Tools Illustrated



7. Conclusion

http://momjian.us/presentations

https://www.flickr.com/photos/eklektikos/