Redundant OpenVPN servers with Yubikey OTP and PIN

v02

Goal: Create a VPN server that authenticates users with:

- SSL certificates stored on the client computer
- physical OTP key that the user carries
- PIN that the user knows

SSL authentication is performed by OpenVPN, the VPN software used for the server and the client.

OTP authentication is handled by Yubico software and hardware. The OTP tokens are USB sticks called Yubikey, that register with the computer as a "keyboard". They generate an OTP every time the sensor is touched by the user.

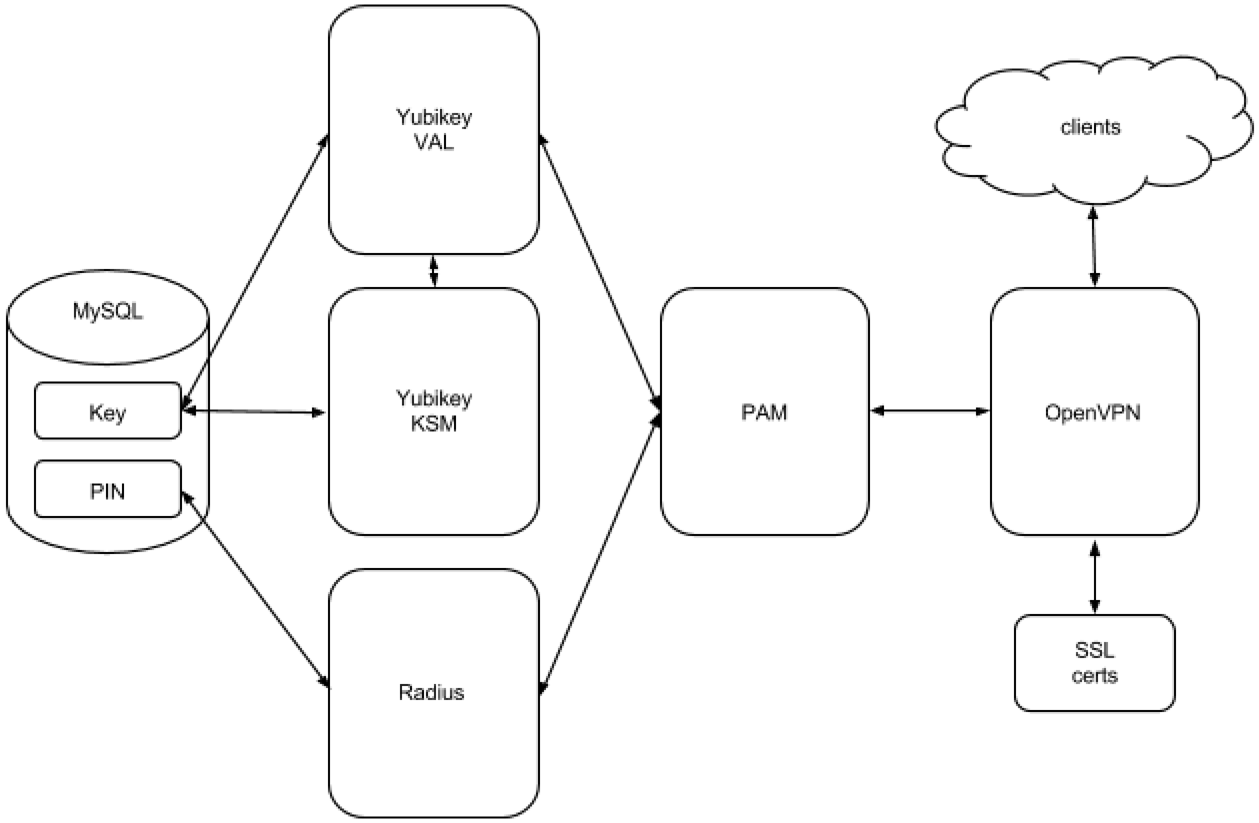
The PIN is stored in a local database. In this particular case, Radius is used to do the PIN authentication.

When the connection is initiated, in the password field the user enters the PIN first. Without pressing Enter, the user then taps the Yubikey, which generates the OTP directly into the password field.

Duplicate the VPN server, establish master/master replication between MySQL on the two servers. Do DNS “load balancing” between the two servers.

Architecture

The internal connections inside a VPN server:



PAM is the authentication broker, with the main configuration file /etc/pam.d/openvpn
Key identities and Radius PINs are stored in MySQL.
/etc/yubico/yubikeyid binds user accounts to key identities.

Making changes to accounts (create, delete, lock, modify) involved making changes to the MySQL database and/or /etc/yubico/yubikeyid

The two VPN servers have their MySQL databases in a master/master replication scheme, via a private VPN tunnel.

Install OpenVPN, configure, test

OpenVPN is a VPN server that uses OpenSSL for encryption. It is not an “SSL VPN”, but a full-fledged VPN software, with complete routing capabilities, point-to-point, LAN-to-LAN, etc. Authentication is typically done with SSL certificates on client and server, but other mechanisms could be used (fixed password, OTP, smart cards / PKCS#11, etc).

Install Ubuntu 14.04 LTS, pretty minimal configuration (ssh is the only service really required for now). Update and reboot.

```
apt-get update
apt-get dist-upgrade
reboot

apt-get install openvpn easy-rsa
useradd -d /var/run/openvpn -m -r -s /usr/sbin/nologin openvpn
```

http://openvpn.net/index.php/open-source/documentation/howto.html

Create your own CA (Certificate Authority)

Once you have your own CA, with a master CA certificate and key, you could generate as many certificates as you wish for servers and clients. This is accomplished with the easy-rsa set of scripts, which uses openssl.

It is important that the location for the CA is **secure** and **unique**. Choose a good place for it, and stick to it. All certificates you generate and revoke are indexed and kept track of within the CA, so don’t start multiple copies of your CA. It cannot be much of an 'authority' if there’s many of them floating around.

Create a directory where you will keep the SSL CA files. Copy all easy-rsa scripts there.

```
cd <secure location>
cp -a /usr/share/easy-rsa .
cd easy-rsa
```

Open the vars file in a text editor and edit the block containing KEY_COUNTRY.

Generate the master CA certificate and key:

```
. vars
```

```
./clean-all
./build-ca
# accept all the defaults here
```

This is what you will have now in the keys subfolder:

- ca.crt
- ca.key
- index.txt
- serial

Generate certificate and key for servers. The common name you choose here (the parameter for the build-key-server script) may reflect the name of the VPN server, or some other name relevant to your context:

```
./build-key-server vpn-yubikey
# accept defaults, leave passwords empty, sign, and commit
```

Generate Diffie-Hellman parameters:

```
./build-dh
```

This is what you have now in keys/:

```
# ls -lh keys/
total 52K
-rw-r--r-- 1 root root 5.6K Sep  3 16:09 01.pem
-rw-r--r-- 1 root root 1.8K Sep  3 16:09 ca.crt
-rw------- 1 root root 1.7K Sep  3 16:09 ca.key
-rw-r--r-- 1 root root  424 Sep  3 16:11 dh2048.pem
-rw-r--r-- 1 root root  140 Sep  3 16:09 index.txt
-rw-r--r-- 1 root root   21 Sep  3 16:09 index.txt.attr
-rw-r--r-- 1 root root    0 Sep  3 16:09 index.txt.old
-rw-r--r-- 1 root root    3 Sep  3 16:09 serial
-rw-r--r-- 1 root root    3 Sep  3 16:09 serial.old
-rw-r--r-- 1 root root 5.6K Sep  3 16:09 vpn-yubikey.crt
-rw-r--r-- 1 root root 1.1K Sep  3 16:09 vpn-yubikey.csr
-rw------- 1 root root 1.7K Sep  3 16:09 vpn-yubikey.key
```

Generate certificate and key for a test client. The name used with this command is the common name of the SSL certificate; it should match the username for the account you're setting up:

```
./build-key jsmith
```

Make a bogus cert and revoke it, just to see how it works:

```
./build-key bogus-cert-please-ignore
./revoke-full bogus-cert-please-ignore
# ignore error 23 at 0 depth lookup:certificate revoked
```

This will create revoke-test.pem in keys/. In case you need to revoke SSL certificates, you will copy that file into /etc/openvpn and declare it in the OpenVPN configuration (see man openvpn). You will have to copy it again every time you revoke another cert.

Create a real certificate for a test user account. The common name for this cert will be the username for the account that will be created, and it will be used everywhere in this document.

```
./build-key jsmith
```

Configure the OpenVPN server

At this point the partial goal is to have a VPN server and a VPN client configured and ready to go, and authenticate them to each other via SSL certificates only. This is pretty typical for simple OpenVPN deployments.

Copy server-related files into the OpenVPN config directory:

```
cd keys
cp ca.crt dh2048.pem vpn-yubikey.crt vpn-yubikey.key /etc/openvpn/
```

Generate the TLS auth key:

```
cd /etc/openvpn
openvpn --genkey --secret ta.key
```

Create a server config file. It can have any name you wish, as long as it ends in .conf. Suggested content:

```
################################
port 1194
proto udp
float
persist-key
#plugin /usr/lib/openvpn/openvpn-plugin-auth-pam.so openvpn
dev tun1
```

```
ca ca.crt
cert vpn-yubikey.crt # edit to match
key vpn-yubikey.key # edit to match
dh dh2048.pem
server 10.114.208.0 255.255.255.0 # edit to match
ifconfig-pool-persist ipp.txt
client-to-client
keepalive 10 120
tls-auth ta.key 0
comp-lzo
user openvpn
group openvpn
persist-key
persist-tun
status openvpn-status.log
verb 3
################################
```

This is the content of /etc/openvpn now:

```
# ls -lh
-rw-r--r-- 1 root root 1.8K Sep  3 16:25 ca.crt
-rw-r--r-- 1 root root  424 Sep  3 16:25 dh2048.pem
-rw------- 1 root root  636 Sep  3 16:35 ta.key
-rwxr-xr-x 1 root root 1.3K Feb  4  2014 update-resolv-conf
-rw-r--r-- 1 root root  451 Sep  3 16:41 vpn-yubikey.conf
-rw-r--r-- 1 root root 5.6K Sep  3 16:25 vpn-yubikey.crt
-rw------- 1 root root 1.7K Sep  3 16:25 vpn-yubikey.key
```

The permissions on the .key files are more restrictive.

Restart the openvpn service, check /var/log/syslog for any surprises.

An OpenVPN server, by definition, forwards traffic between clients and connection end-points. By default, IP forwarding is disabled on most distributions, including Ubuntu:

```
# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
```

To enable it, create /etc/sysctl.d/99-openvpn.conf with the contents:

```
net.ipv4.ip_forward=1
```

Start procps and check the variable again:

```
# service procps start
procps stop/waiting
# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

You should also verify that the iptables FORWARD chain does not block traffic - but this will be discussed later in the chapter concerning security. For now, just make sure the FORWARD chain is empty and has the ACCEPT policy (which is essentially always the case on a fresh install):

```
# iptables -L FORWARD -nv
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
destination
#
```

If the FORWARD chain doesn't look like that, flush it and reset the policy:

```
iptables -t filter -F FORWARD
iptables -t filter -P FORWARD ACCEPT
```

This has security implications, so make sure you understand what you're doing.

Finally, iptables configurations can be made persistent, so your changes might be overwritten after reboot. See this document for some suggestions:

https://help.ubuntu.com/community/IptablesHowTo

Again, the discussion on security at the end is relevant for this topic.

Configure the OpenVPN client and test it

Now make a .zip bundle with all files needed for the VPN client. First create an empty folder:

```
cd
mkdir jsmith
cd jsmith
```

Create an OpenVPN config file for the client. The extension .ovpn is preferred by some clients. The name can be anything, but ideally it should match the name of account being created. This is the content:

```
###################################
client
dev tun
persist-tun
#auth-user-pass
#auth-retry interact
proto udp
remote vpn-yubikey.yourdomain.com 1194 #edit to match server
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
cert jsmith.crt #edit to match account
key jsmith.key #edit to match account
ns-cert-type server
tls-auth ta.key 1
comp-lzo
verb 3
###################################
```

Copy the cert and key for client and the CA cert from the secure location, and the TLS auth key from the server config dir, to the current dir:

```
cp <secure-location>/keys/jsmith.crt .
cp <secure-location>/keys/jsmith.key .
cp <secure-location>/keys/ca.crt .
cp /etc/openvpn/ta.key .
```

These are all files needed for the client:

```
# ls -l
total 24
-rw-r--r-- 1 root root 1749 Sep  3 17:03 ca.crt
-rw-r--r-- 1 root root 5493 Sep  3 17:01 jsmith.crt
-rw------- 1 root root 1704 Sep  3 17:01 jsmith.key
-rw-r--r-- 1 root root  301 Sep  3 17:00 jsmith.ovpn
-rw------- 1 root root  636 Sep  3 17:03 ta.key
```

Make the archive for the new client:

```
cd ..
zip -r jsmith.zip jsmith/
```

Installing the client files depends on the client you're using. E.g., for the Tunnelblick OpenVPN client for OS X:

- unzip archive
- in Finder, double-click the .ovpn file and import configuration
- Install Configuration For All Users? - "Only Me"
- click Tunnelblick icon in the task bar, click VPN Details
- select the new configuration you've created
- Connect: "Manually"
- option “Set DNS/WINS” change to “Do not set nameserver”
- select “Keep connected”

Now connect to the server, while watching /var/log/syslog on the server.
If successful, from the client ping the IP address of the server’s tun1 interface - it should only be reachable when the tunnel is connected.

Install the Yubico software

To use OTP authentication with the Yubikey devices, you have to install the library (and optionally servers) that can handle OTP.

Yubico has an Ubuntu repository with binaries that contains everything you need:

```
add-apt-repository ppa:yubico/stable
apt-get update
apt-get install yubikey-ksm yubikey-val libpam-yubico
```

The yubikey-ksm and yubikey-val packages are the local authentication servers. You only need them if you want to create your own local OTP authentication backend.

If you just want to use the Yubico public authentication cloud with your key, then ignore these two packages and install only the PAM library module. In that case, the MySQL server dependency will be installed later (when FreeRadius is installed).

While installing the yubikey packages, you'll be prompted to perform a few tasks:

- create a root password for MySQL and write it down
- configure database for yubikey-ksm; choose type mysql; leave password blank (will be - created automatically
- same for yubikey-val

Stop and disable the ykval-queue service (it's not needed, and it's very verbose in the logs):

```
service ykval-queue stop
update-rc.d ykval-queue disable
```

http://blog.bogosity.se/2014/04/13/requering-both-an-ssh-key-and-a-yubikey/

http://www.raczylo.com/blog/OpenVPN-with-YubiKey-and-GoogleAuthenticator.html

http://pragmasec.wordpress.com/2014/07/12/set-up-yubikey-with-pam-for-openvpn-ssh-and-squirrelmail/

Test OpenVPN authentication with the public Yubico cloud

The OpenVPN infrastructure is authenticating already with SSL client/server certificates. It is time now to add OTP authentication on top of that. OpenVPN can use PAM as an authentication broker for a very wide variety of authentication backends (Unix passwords, etc). It is now a simple matter of adding Yubico as a backend to PAM on the OpenVPN server.

Edit the OpenVPN server config file /etc/openvpn/vpn-yubikey.conf and uncomment the line below, then restart the openvpn service:

```
plugin /usr/lib/openvpn/openvpn-plugin-auth-pam.so openvpn
```

The “openvpn” parameter at the end tells OpenVPN to use for its own authentication the PAM statements that we will put into /etc/pam.d/openvpn below.

Create /etc/pam.d/openvpn with the contents:

```
auth  required   pam_yubico.so authfile=/etc/yubico/yubikeyid id=16
debug
account    required   pam_permit.so debug
```

The first line tells PAM to use the pam_yubico.so module with its default authentication backend (the public Yubico auth cloud) using client ID 16, and map user identities to key identities using the dictionary in /etc/yubico/yubikeyid.

The second line tells PAM to not verify whether it should check that the Unix account for the authenticated used is actually created (we store all used info in the database).

Insert the Yubikey into a USB port and launch a text editor. If it's a Neo key, do a short press (tap) on the Yubikey sensor; for older keys do a normal press (it doesn't matter). The OTP will be printed in the text editor, prefixed by 12 characters which are the ID of this Yubikey (or the ID of slot 1, if it's a multi-slot like the Neo).

Copy the 12-character ID from the head of the printed string. Create the file /etc/yubico/yubikeyid and add the test username and the key ID to it in the following format:

```
username:key-id
```

E.g.:

```
jsmith:cccccbhkljr
```

The username here is the common name used when the client VPN certificate was created (basically, the name of this user account which will be used everywhere in this HOWTO as an example).

In the OpenVPN client configuration file jsmith.ovpn remove comments from these lines:

```
auth-user-pass
auth-retry interact
```

For the Tunnelblick VPN client, the config files are located in /Library/Application\ Support/Tunnelblick/Shared (if shared) or ~/Library/Application\ Support/Tunnelblick/Configurations (if private).

https://code.google.com/p/tunnelblick/wiki/cFileLocations#Configuration_Files

Now try to connect to the VPN server. You will be prompted for username and password. For the password field, click it and then short-press the Yubikey sensor - the OTP will fill that field automatically.

If all goes well, you'll connect to the server. From the client, ping the tun interface of the server to prove that the tunnel is up.

Install and configure Freeradius

Radius will be used to verify the fixed password for the user account, which will be used in addition to the OTP, the way a PIN is used with a debit card. The reason why Radius was chosen here is simplicity: it's an easy way to store the PIN in a database, and then query it. Also, authenticating PAM against Radius is widely used and pretty robust.

Install FreeRadius:

```
apt-get install freeradius freeradius-mysql apg libpam-radius-auth
```

http://techtots.blogspot.com/2010/01/installing-and-configuring-freeradius.html

http://wiki.freeradius.org/guide/SQL-HOWTO

https://extremeshok.com/2197/ubuntu-12-04-12-10-install-freeradius-server-authenticating-with-a-mysql-mariadb-database/

Run apg to generate a random password for the radius user (will be used by freeradius to connect to the DB). Create radius DB and user:

```
CREATE DATABASE radius;
GRANT ALL ON radius.* TO radius@localhost IDENTIFIED BY "PASSWORD";
```

Edit /etc/freeradius/sql.conf and add to it the password you've generated.

Now create the DB schema:

```
mysql -uroot -p radius < /etc/freeradius/sql/mysql/schema.sql
```

To enable the radius SQL backend, edit /etc/freeradius/radiusd.conf and uncomment this line:

```
$INCLUDE sql.conf
```

In the log{} section, make these changes:

```
destination = syslog
auth = yes
```

Edit /etc/freeradius/sites-available/default and uncomment sql in the authorize{} section.

Edit /etc/freeradius/clients.conf and change the radius secret to a more secure value (use the apg utility to generate random strings).

Now it's time to add your VPN test account to the Radius database, along with its PIN. In the example below I've chosen the PIN "1234":

```
INSERT INTO radius.radcheck (username, attribute, op, value) VALUES
('jsmith', 'MD5-Password', ':=', MD5('1234'));
```

Restart the Radius server:

```
service freeradius restart
```

Sometimes this service gets stuck when you restart it the first time after it's installed - check /var/log/syslog. If so, then do a "service freeradius stop", kill the remaining freeradius process manually, then start the service again. After that you should have no problems restarting it.

Test it with the command-line tool:

```
# radtest jsmith 1234 localhost:1812 1 <radius-secret>
Sending Access-Request of id 225 to 127.0.0.1 port 1812
     User-Name = "jsmith"
     User-Password = "1234"
     NAS-IP-Address = 127.0.1.1
     NAS-Port = 1
     Message-Authenticator = 0x00000000000000000000000000000000
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=225,
length=20
```

If you get Access-Accept then it’s fine. Check /var/log/syslog for troubleshooting.

Note: Currently, during the boot-up sequence, the Ubuntu freeradius package starts the service too soon, before mysql is running. Because of that, freeradius fails to connect to the DB. As a workaround, I restart free freeradius later during boot, by adding this line to /etc/rc.local:

```
(sleep 20; service freeradius stop; sleep 1; service freeradius
start) &
```

Test authentication with OTP (public Yubico auth cloud) + local Radius PIN

Time to add the Radius PIN to the existing authentication mechanisms on the VPN server. This is accomplished by stacking up various PAM modules - specifically, by inserting the Radius PAM module in the existing stack in /etc/pam.d/openvpn.

To configure the Radius PAM module, edit /etc/pam_radius_auth.conf and add a line containing localhost, with the secret you've entered in /etc/freeradius/clients.cfg. This will tell PAM to use localhost as a Radius authentication server.

```
127.0.0.1  your-secret     3
```

The comments in that file tell you to copy it somewhere else - don’t do that if you do the standard Ubuntu installation from repos.

Now edit /etc/pam.d/openvpn and insert a line to enable Radius authentication. The file should now look like this:

```
auth  required   pam_yubico.so authfile=/etc/yubico/yubikeyid id=16
debug
auth  required   pam_radius_auth.so debug
account    required   pam_permit.so debug
```

The order is important. The Yubico module must process the PIN + OTP string first, since it knows how to remove the OTP from it before handing it down to the next module.

After that, the Radius module verifies the PIN against the database.

The line with pam_permit.so at the end is there to allow you to create accounts only in the database, with no need to also create Unix accounts on the OS. If you omit that line, authentication will still go to the OTP servers and to Radius, but it will ultimately fail anyway because the PAM stack also checks the existence of the Unix account, if you don’t force a successful return from the account existence check.

https://www.wikidsystems.com/support/wikid-support-center/how-to/how-to-configure-pam-radius-in-ubuntu

https://www.digitalocean.com/community/tutorials/how-to-use-pam-to-configure-authentication-on-an-ubuntu-12-04-vps

Also see all the documentation files in /usr/share/doc/libpam-yubico, installed with the libpam-yubico package.

On the VPN server, begin watching /var/log/syslog with tail -f. No need to restart anything.

Plug the Yubikey in your VPN client computer and connect the test VPN tunnel. At the prompt, enter your test username. Move the cursor to the password field and enter the Radius PIN. **Do not press Enter** after that. Just leave it as is, and touch the sensor on the Yubikey - it will print the OTP into the password field and it will “hit Enter” automatically at the end.

The idea is that the “password” you send to the VPN server is the Radius PIN concatenated with the Yubico OTP. PAM will extract the various parts and treat them separately.

If all goes well, the tunnel will be established. In the logs, you’ll see something similar to this:

```
Sep  5 10:53:06 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 TLS: Initial packet from
[AF_INET]172.28.128.1:58543, sid=3a90df63 09e37994
Sep  5 10:53:06 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 VERIFY OK: depth=1, C=US, ST=CA, L=MyHomeTown,
O=thiscompany, OU=Engineering, CN=thiscompany CA, name=EasyRSA,
emailAddress=eng@thiscompany.com
Sep  5 10:53:06 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 VERIFY OK: depth=0, C=US, ST=CA, L=MyHomeTown,
O=thiscompany, OU=Engineering, CN=jsmith, name=EasyRSA,
emailAddress=eng@thiscompany.com
Sep  5 10:53:11 vbox-ubuntu-1404 openvpn[1027]: pam_radius_auth:
DEBUG: getservbyname(radius, udp) returned 1048382528.
Sep  5 10:53:11 vbox-ubuntu-1404 freeradius[1808]: Login OK: [jsmith]
(from client localhost port 1027)
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 PLUGIN_CALL: POST
/usr/lib/openvpn/openvpn-plugin-auth-pam.so/PLUGIN_AUTH_USER_PASS_VER
IFY status=0
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 TLS: Username/Password authentication succeeded
for username 'jsmith'
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 Data Channel Encrypt: Cipher 'BF-CBC' initialized
with 128 bit key
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 Data Channel Encrypt: Using 160 bit message hash
'SHA1' for HMAC authentication
```

```
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 Data Channel Decrypt: Cipher 'BF-CBC' initialized
with 128 bit key
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 Data Channel Decrypt: Using 160 bit message hash
'SHA1' for HMAC authentication
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 Control Channel: TLSv1, cipher TLSv1/SSLv3
DHE-RSA-AES256-SHA, 2048 bit RSA
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
172.28.128.1:58543 [jsmith] Peer Connection Initiated with
[AF_INET]172.28.128.1:58543
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 MULTI_sva: pool returned IPv4=10.114.208.6,
IPv6=(Not enabled)
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 MULTI: Learn: 10.114.208.6 ->
jsmith/172.28.128.1:58543
Sep  5 10:53:11 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 MULTI: primary virtual IP for
jsmith/172.28.128.1:58543: 10.114.208.6
Sep  5 10:53:14 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 PUSH: Received control message:
'PUSH_REQUEST'
Sep  5 10:53:14 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 send_push_reply(): safe_cap=940
Sep  5 10:53:14 vbox-ubuntu-1404 ovpn-vpn-yubikey[1034]:
jsmith/172.28.128.1:58543 SENT CONTROL [jsmith]: 'PUSH_REPLY,route
10.114.208.0 255.255.255.0,topology net30,ping 10,ping-restart
120,ifconfig 10.114.208.6 10.114.208.5' (status=1)
```

At this point you have a fully functional VPN server, authenticating with:

- SSL certificates in the VPN user profile, stored on your laptop / desktop in the VPN client
- Radius PIN, stored in the user’s brain
- OTP generated by the Yubikey

You’re authenticating against the public Yubico auth cloud, and there is no redundancy for the VPN server. If this is what you need, you could stop now.

The following chapters describe how to implement your own Yubico authentication servers locally, and how to build a pair of redundant VPN servers.

Generate a key identity and store it in your local Yubico auth servers

To use your own local authentication servers, instead of the public Yubico cloud, you need to store the whole identity of your Yubikey in your servers. Since it’s impossible to extract the private ID string and secret key from an existing Yubikey device, you have to generate an ID in software, and commit it to your Yubikey.

If you have a Yubikey Neo, this key has two slots, each slot with a separate identity. Slot 1 is activated by a short press (tap) on the sensor, slot 2 is activated by a long press (about 3 seconds). I believe slot 2 is empty by default. You could keep the factory default identity on slot 1 unchanged, and overwrite only slot 2. Then you could authenticate that Yubikey Neo against your servers with a long press (slot 2), while the short press (slot 1) remains available for when you want to authenticate against the public Yubico cloud for some other service.

If you have a Yubikey Standard or Nano, these keys have only one slot. You will have to overwrite the only existing identity on it. If this key is already tied into authenticating another service against the public Yubico cloud, **the key will stop working** there. You have been warned.

To recap, the identity can not be fully extracted from a key entirely (including the secrets), but it could be overwritten with a new identity. It is possible to password-protect the key so ID overwrites in the future can only be done by you, but it’s best to not enable that feature before you’re very familiar with the hardware. By default, the key is not password-protected (at least my Neo wasn’t).

This chapter and the following are largely based on this HOWTO:

http://forum.yubico.com/viewtopic.php?f=31&t=1424

For the next steps you’ll need gpg. If it’s not installed, then:

```
apt-get install gnupg
```

You’ll also need the ykksm-gen-keys and ykksm-import tools. These executables should be contained in the yubikey-ksm package installed a few pages before.

On Ubuntu, everything is contained in the APT repository. You don't need to download and compile anything. But if you need to perform these steps on a system where ykksm-gen-keys is not available in a package (like on OS X currently), then get the software tarballs at this link (not necessary on Ubuntu):

https://developers.yubico.com/yubikey-ksm/

The tools are just Perl scripts, so they should work almost anywhere.

The next steps will perform a lot of crypto. Make sure your system has plenty of entropy. If various commands seem to get stuck, then move the mouse around, tap the CTRL key, generate some hard drive activity, or some network traffic - all these replenish entropy and your tools should get unstuck eventually. If you’re on a headless server, this can be pretty difficult to accomplish - but sometimes a ping flood will crank the entropy up.

First generate the GPG import key:

```
$ gpg --gen-key
gpg (GnuPG) 1.4.18; Copyright (C) 2014 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/Users/someuser/.gnupg' created
gpg: new configuration file `/Users/someuser/.gnupg/gpg.conf' created
gpg: WARNING: options in `/Users/someuser/.gnupg/gpg.conf' are not
yet active during this run
gpg: keyring `/Users/someuser/.gnupg/secring.gpg' created
gpg: keyring `/Users/someuser/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 2
DSA keys may be between 1024 and 3072 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y
```

```
You need a user ID to identify your key; the software constructs the
user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: yubikey-ksm import key
Email address:
Comment:
You selected this USER-ID:
    "yubikey-ksm import key"

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

We need to generate a lot of random bytes. It is a good idea to
perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: WARNING: some OpenPGP programs can't handle a DSA key with this
digest size
+++++....++++++++++++++++++++..++++++++++...+++++++++++++++++++++++++
..+++++++++++++++..+++++.+++++++++++++++++++++++++++++++++++...++++++
++++.++++++++++>+++++..........+++++
We need to generate a lot of random bytes. It is a good idea to
perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
.++++++++++.++++++++++++++++++++++++++++++..+++++.++++++++++..+++++++
+++++++++++++.++++++++++.+++++.+++++.+++++.++++++++++++++++++++.+++++
++++++++++++++++++++>.++++++++++.......>+++++........................
.....................................................................
.....................................................................
......................+++++^^^^^
gpg: /Users/someuser/.gnupg/trustdb.gpg: trustdb created
gpg: key C14E5A21 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048D/C14E5A21 2014-09-05
```

```
      Key fingerprint =
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
uid                  yubikey-ksm import key
sub   2048g/6F539F90 2014-09-05
```

Use a strong passphrase for this key. It's best to keep the whole contents of the .gnupg directory in some safe place. For the real name, use something like "yubikey-ksm import key" or whatever makes sense to you.

Write down the public fingerprint of the key (the line beginning with "pub" above), or you could always get it later with gpg --list-keys. The results of the command you've run are in .gnupg in your home directory.

Now you'll generate the new key identity. Ideally, this should be done on a secure system, with all data piped between commands directly via shell pipes, so the identity is never stored on long term storage. But here we will store it temporarily in text files, for clarity.

For the serial number of the key, pick some random number. The Unix time in seconds might be a good option, since perfect randomness is not critical at this step.

```
ks=`date +%s`
ykksm-gen-keys $ks > key${ks}.txt
```

This is the result:

```
$ cat key1409952015.txt
# ykksm 1
#
serialnr,identity,internaluid,aeskey,lockpw,created,accessed[,progfla
gs]
1409952015,ccccgfcldkcv,9f44acc75b0f,5706256dd94fe0be73b32cecef4fae40
,769819cbca50,2014-09-05T14:20:48,
# the end
```

The long line with lots of numbers has several fields:

- the first field (14099...) is the serial number you used when you generated the key
- the second field (cccc...) is the public ID of the key - this is an important field that you will use later
- the third field (9f44...) is the private ID
- the fourth field (5706...), which is the longest, is the secret key

Now ASCII-armor that file using the GPG key generated above. For the -r parameter use the fingerprint of the GPG key (see above).

```
$ gpg -a --encrypt -r C14E5A21 -s key1409952015.txt

You need a passphrase to unlock the secret key for
user: "yubikey-ksm import key"
2048-bit DSA key, ID C14E5A21, created 2014-09-05
```

This is what you get:

```
$ ls -lh key1409952015.txt*
-rw-r--r--  1 florinandrei  staff   199B Sep  5 14:20
key1409952015.txt
-rw-r--r--  1 florinandrei  staff   1.3K Sep  5 14:25
key1409952015.txt.asc
```

Import this identity into the KSM (Key Store Manager) database. The database name, username and password you need at this step are in /etc/yubico/ksm/config-db.cfg. You will also be prompted for the passphrase for the GPG import key:

```
ykksm-import --verbose --database
'DBI:mysql:dbname=ykksm;host=127.0.0.1' --db-user ykksmreader
--db-passwd <ksm-db-pass> < ./key1409952015.txt.asc

[...snip...]

You need a passphrase to unlock the secret key for
user: "yubikey-ksm import key"
2048-bit ELG-E key, ID 6F539F90, created 2014-09-05 (main key ID
C14E5A21)

line:
1409952015,ccccgfcldkcv,9f44acc75b0f,5706256dd94fe0be73b32cecef4fae40
,769819cbca50,2014-09-05T14:20:48,
     serialnr 1409952015 publicname ccccgfcldkcv internalname
9f44acc75b0f aeskey 5706256dd94fe0be73b32cecef4fae40 lockcode
769819cbca50 created 2014-09-05T14:20:48 accessed  eol
```

Again, you can see here in the output the public ID, the private ID, and the secret key.

Copy the public ID (in this case cccc...) Create a file named /etc/yubico/yubikeyid and add the public key ID to the username that will use this key. It's okay for the same username to have more than one key ID, although in practice most users will probably have just one key.

Basically, this file connects the key ID to the identity of the user. Each username is on a separate line.

```
# cat /etc/yubico/yubikeyid
jsmith:cccccbhkljr:ccccgfcldkcv
```

Check the database, the key should be stored there:

```
SELECT * FROM ykksm.yubikeys;
```

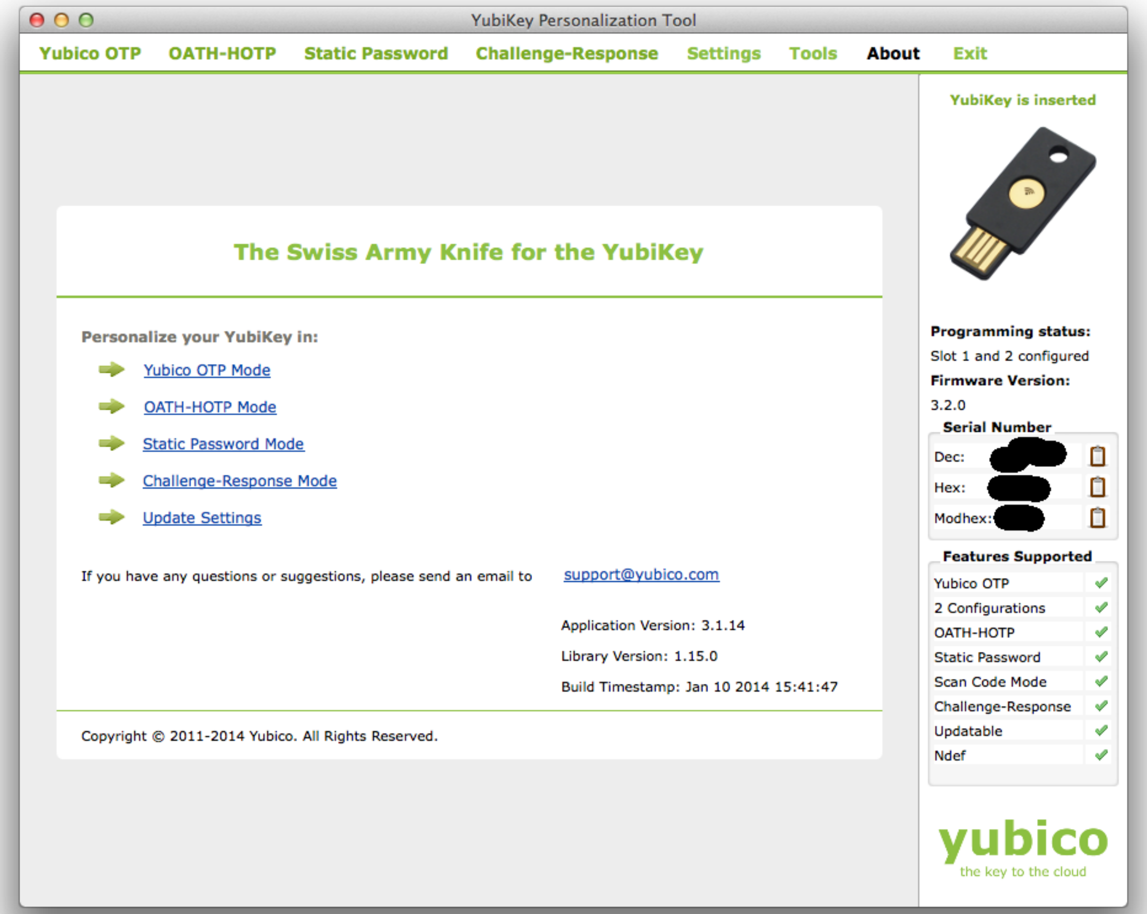
Compare the database with the data about the key stored in the text file, it should be the same.

Upload the key ID into a Yubikey device

To use the key ID you've created and stored, you have to upload it into an actual Yubikey device. First, install the YubiKey Cross Platform Personalization Tool, which has binaries and packages for most platforms:

http://www.yubico.com/products/services-software/personalization-tools/use/

Plug the Yubikey into your computer. Then launch the personalization tool.



The Swiss Army Knife for the YubiKey

Personalize your YubiKey in:

- Yubico OTP Mode
- OATH-HOTP Mode
- Static Password Mode
- Challenge-Response Mode
- Update Settings

If you have any questions or suggestions, please send an email to support@yubico.com

Application Version: 3.1.14
Library Version: 1.15.0
Build Timestamp: Jan 10 2014 15:41:47

Copyright © 2011-2014 Yubico. All Rights Reserved.

Programming status:
Slot 1 and 2 configured
Firmware Version:
3.2.0

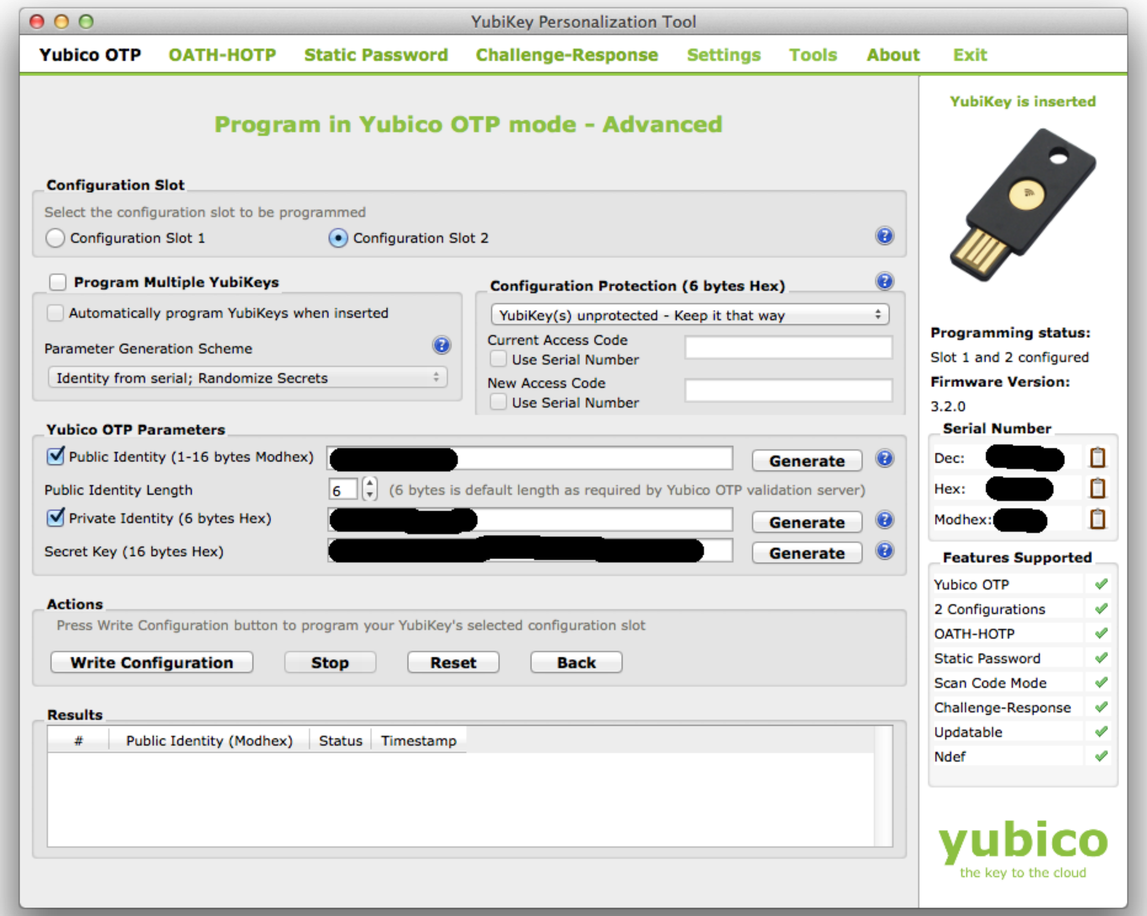
Serial Number

Dec:	[redacted]
Hex:	[redacted]
Modhex:	[redacted]

Features Supported

Yubico OTP	✓
2 Configurations	✓
OATH-HOTP	✓
Static Password	✓
Scan Code Mode	✓
Challenge-Response	✓
Updatable	✓
Ndef	✓

Click the Yubico OTP Mode link to program the Yubikey. Then select Advanced. You will then see this screen:


YubiKey Personalization Tool

Yubico OTP | OATH-HOTP | Static Password | Challenge-Response | Settings | Tools | About | Exit

Program in Yubico OTP mode - Advanced

Configuration Slot
Select the configuration slot to be programmed
( ) Configuration Slot 1 [● Configuration Slot 2]

[ ] **Program Multiple YubiKeys**
[ ] Automatically program YubiKeys when inserted
Parameter Generation Scheme: Identity from serial; Randomize Secrets

Configuration Protection (6 bytes Hex)
YubiKey(s) unprotected - Keep it that way
Current Access Code: [ ] Use Serial Number
New Access Code: [ ] Use Serial Number

Yubico OTP Parameters
[✓] Public Identity (1-16 bytes Modhex): [redacted] Generate
Public Identity Length: 6 (6 bytes is default length as required by Yubico OTP validation server)
[✓] Private Identity (6 bytes Hex): [redacted] Generate
Secret Key (16 bytes Hex): [redacted] Generate

Actions
Press Write Configuration button to program your YubiKey's selected configuration slot
Write Configuration | Stop | Reset | Back

Results

#	Public Identity (Modhex)	Status	Timestamp

YubiKey is inserted

Programming status: Slot 1 and 2 configured
Firmware Version: 3.2.0

Serial Number
Dec: [redacted]
Hex: [redacted]
Modhex: [redacted]

Features Supported

Yubico OTP	✓
2 Configurations	✓
OATH-HOTP	✓
Static Password	✓
Scan Code Mode	✓
Challenge-Response	✓
Updatable	✓
Ndef	✓

yubico the key to the cloud

Select Configuration Slot 2 if you use a Neo. Other keys may only have one slot (see discussion in previous chapter).

Now refer to the file key1409952015.txt generated above. The fields in that file need to be copied to the GUI: public ID (identity), private ID (internaluid), and secret key (aeskey), highlighted in bold characters below:

```
$ cat key1409952015.txt
# ykksm 1
#
serialnr,identity,internaluid,aeskey,lockpw,created,accessed[,progfla
gs]
1409952015,ccccgfcldkcv,9f44acc75b0f,5706256dd94fe0be73b32cecef4fae40
,769819cbca50,2014-09-05T14:20:48,
# the end
```

The section titled Yubico OTP Parameters in the screenshot above needs to receive these fields (the three big blacked out parts in the middle of the screenshot). Copy each field and paste it into the corresponding place in the GUI. Then click Write Configuration.

You could write the output to a log file. If all goes well, the GUI will print a status message along the lines of "YubiKey has been successfully configured".

Close the GUI. Start a text editor. Do a long-press on the Yubikey sensor (if you've programmed slot 2 on a Neo), or a short press or tap (if you've programmed slot 1 on a Neo, or if you've programmed one of the other, simpler keys). Watch the text editor.

A long string of text will be printed into the editor. The first 12 characters will be the public ID of the identity you've just programmed into your Yubikey (cccc....). The rest is the OTP. Make sure the public ID you get from the key actually matches what you think you've uploaded into it.

Configure the OpenVPN server to use the local authentication database for OTP

Now it's time to switch your authentication from using the public Yubico OTP cloud to your own local authenticator.

http://www.andybotting.com/using-the-yubikey-for-two-factor-authentication-on-linux

All you need to do is edit /etc/pam.d/openvpn and point it at the local authenticator. You could keep the old entry pointing it at the public cloud, if you like, just make sure to comment it out.

In the default installation of the Yubikey VAL server, the only client ID in the database is 1. So you'll have to adjust the id= parameter for the local authentication. In the end, /etc/pam.d/openvpn should look like this:

```
# public OTP authentication
#auth required   pam_yubico.so authfile=/etc/yubico/yubikeyid id=16
debug
# local OTP authentication
auth  required   pam_yubico.so authfile=/etc/yubico/yubikeyid id=1
url=http://127.0.0.1/wsapi/2.0/verify?id=%d&otp=%s debug
#
# Radius PIN
auth  required   pam_radius_auth.so debug
#
# Avoid having to create local Unix accounts
account    required   pam_permit.so debug
```

http://127.0.0.1/ in the file above indicates you're using localhost for OTP authentication. In theory you could run the VAL server somewhere else, and then you'd have to change that URL, but that's another story for another day.

Now go ahead and test it. "tail -f /var/log/syslog" on the server. No need to restart anything.

Fire up the VPN client and wait for the user/pass prompt. Enter the username, then click the password field.

In the password field, enter the Radius PIN, but do not hit Enter yet - instead, on the Yubikey do a long-press (if it's a Neo with slot 2 programmed), or a short press (if it's a Neo with slot 1 programmed, or one of the other keys with only one slot). You will see the Yubikey filling up the password field with characters.

If all goes well, the VPN client will connect; on the server, in syslog you'll see a ton of messages from openvpn, radius and ykval indicating success.

In the database, after each successful authentication, do this query:

```
SELECT * FROM ykval.yubikeys;
```

The following fields will change each time you authenticate:

- modified
- yk_use
- yk_low
- yk_high

If you don't need redundant VPN servers, you could stop here.

Create a pair of redundant VPN servers

Most of the authentication information is contained in the MySQL database. There are very few things that are outside it, that need to be maintained as you add / remove users. The glaring exception is the /etc/yubico/yubikeyid file that maps Yubikey IDs to usernames - that one will have to be copied somehow between your multiple VPN servers (rsync?).

But for all the rest, DB replication will take care of it.

Generic configuration

First, make sure both servers have the same software installed. You could simply go on the secondary server and run all the apt-get commands that you've run on the primary.

```
apt-get update
apt-get dist-upgrade
reboot

add-apt-repository ppa:yubico/stable
apt-get update
apt-get install openvpn easy-rsa yubikey-ksm yubikey-val
libpam-yubico freeradius freeradius-mysql apg libpam-radius-auth
gnupg
```

Use the same passwords above that you've used for the primary server. The files on the primary with the passwords are:

- /etc/yubico/ksm/config-db.cfg
- /etc/yubico/ksm/config-db.php
- /etc/yubico/val/config-db.php

Perform various configuration tasks:

```
update-rc.d ykval-queue disable
service ykval-queue stop
useradd -d /var/run/openvpn -m -r -s /usr/sbin/nologin openvpn
echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/99-openvpn.conf
service procps start
```

In the middle of /etc/rc.local (not at the end) insert this:

```
(sleep 20; service freeradius stop; sleep 1; service freeradius
start) &
```

To be honest, you could run the above sequence even when building the primary - just get all the software installed in one shot.

Now make sure the primary and secondary have the same configuration. Everything in the following locations must be identical (provided that the DB passwords for YK are made identical):

- /etc/yubico
- /etc/openvpn
- /etc/freeradius
- /etc/pam.d/openvpn
- /etc/pam_radius_auth.conf

If you haven't chosen identical DB passwords (between primary and secondary) for the YK software, and you change your mind later, this is how to change them in the DB:

```
SET PASSWORD FOR ykksmreader@localhost = PASSWORD('ksm-password');
SET PASSWORD FOR ykval_verifier@localhost = PASSWORD('val-password');
```

The following steps assume you have two VPN servers: **primary** and **secondary**.

On the secondary, create the Radius DB, with schema and user:

```
CREATE DATABASE radius;
GRANT ALL ON radius.* TO radius@localhost IDENTIFIED BY
"the-radius-db-password";

mysql -uroot -p radius < /etc/freeradius/sql/mysql/schema.sql
```

If passwords are identical between the two servers, then **on the primary** you could just tar up all files that are supposed to be identical...

```
tar -czvf all.tar.gz /etc/yubico /etc/openvpn /etc/freeradius
/etc/pam.d/openvpn /etc/pam_radius_auth.conf
```

...and untar the whole thing **on the secondary** in / then reboot.

```
cd /
tar -zxvf /path/to/all.tar.gz
reboot
```

Check /var/log/syslog for any errors.

Setup MySQL master/master replication

Master/master replication means both database servers are masters, and both are slaves. In other words, you could write data to either server, and read from either server; data is replicated between servers in either direction.

You should give some thought to the connection between servers. If both servers are on the same LAN, you could just replicate directly from one IP to another.

If they are in separate locations, especially if separated by insecure networks, you must create a dedicated point-to-point VPN tunnel between the two systems, and setup replication over the tunnel. This is outside the scope of this document, but should be pretty trivial to implement, if you're familiar with OpenVPN.

In any case, on each server you must pick an IP address that will be used for replication. Often, that's the only IP address the server has; sometimes a separate interface (physical or virtual) is dedicated for replication on each machine. In the example below, the IP addresses we use are the replication IPs on the two servers - the procedure is the same no matter whether the IPs are dedicated or not.

On the primary server, edit /etc/mysql/conf.d/replication.cnf with the following content:

```
[mysqld]
server-id        = 1
log_bin          = /var/log/mysql/mysql-bin.log
binlog_do_db     = radius
binlog_do_db     = ykksm
binlog_do_db     = ykval
bind-address     = *
```

And restart the mysql service (service mysql restart).

Create the replication user and grant it privileges:

```
CREATE USER replicator@'%' IDENTIFIED BY '<rep-pass>';
GRANT REPLICATION SLAVE ON *.* TO replicator@'%';
```

Dump all replicated databases:

```
mysqldump -p --databases radius ykksm ykval > pri.sql
```

Verify the master status:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------------+-----------------
-+
| File             | Position | Binlog_Do_DB       | Binlog_Ignore_DB
|
+------------------+----------+--------------------+-----------------
-+
| mysql-bin.000001 |      344 | radius,ykksm,ykval |
|
+------------------+----------+--------------------+-----------------
-+
1 row in set (0.00 sec)
```

On the secondary execute the following:

Import the databases from the dump file:

```
cat pri.sql | mysql -p
```

Edit /etc/mysql/conf.d/replication.cnf with:

```
[mysqld]
server-id       = 2
log_bin         = /var/log/mysql/mysql-bin.log
binlog_do_db    = radius
binlog_do_db    = ykksm
binlog_do_db    = ykval
bind-address    = *
```

Restart mysql (service mysql restart).

Create the replication user and grant it privileges:

```
CREATE USER replicator@'%' IDENTIFIED BY '<repl-pass>';
GRANT REPLICATION SLAVE ON *.* TO replicator@'%';
```

Stop the slave:

```
SLAVE STOP;
```

Give the secondary all the info it needs about the master: the IP of the primary (that is used for replication), user and pass, the log file, and the position in the log. All this information was gathered above when running SHOW MASTER STATUS on the primary. The IP address is shown by ifconfig.

```
CHANGE MASTER TO MASTER_HOST = '172.28.128.215', MASTER_USER =
'replicator', MASTER_PASSWORD = '<repl-pass>', MASTER_LOG_FILE =
'mysql-bin.000001', MASTER_LOG_POS = 344, MASTER_CONNECT_RETRY = 10;
```

And start the slave:

```
SLAVE START;
```

Finally, show the master status:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------------+-----------------
-+
| File             | Position | Binlog_Do_DB       | Binlog_Ignore_DB
|
+------------------+----------+--------------------+-----------------
-+
| mysql-bin.000001 |      378 | radius,ykksm,ykval |
|
+------------------+----------+--------------------+-----------------
-+
1 row in set (0.00 sec)
```

On the primary now execute these steps:

Stop the slave:

```
SLAVE STOP;
```

Again on the primary, tell it to use the secondary as a master, and give it the IP of the secondary, the user/pass, and the log file name and position. This information was provided above when you did "show master status" on the secondary. The IP address is the secondary's replication interface.

```
CHANGE MASTER TO MASTER_HOST = '172.28.128.216', MASTER_USER =
'replicator', MASTER_PASSWORD = '<repl-pass>', MASTER_LOG_FILE =
'mysql-bin.000001', MASTER_LOG_POS = 344, MASTER_CONNECT_RETRY = 10;
```

Start the slave on the primary:

```
SLAVE START;
```

Now master/master replication should be active. Test it:

On both servers, use the same database:

```
USE radius;
```

On one server, create a dummy table:

```
CREATE TABLE dummy (`id` varchar(10));
```

On the other server, try to see that table:

```
mysql> DESCRIBE dummy;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

Success!

On the server where you've run DESCRIBE, try to drop that table:

```
DROP TABLE dummy;
```

And back on the server where you've create it, try to see if that table still exists:

```
mysql> DESCRIBE dummy;
ERROR 1146 (42S02): Table 'radius.dummy' doesn't exist
```

Perfect! Test the other replicated databases too, to make sure they are being replicated. At this point you have full replication between servers.

Connect VPN to the primary server. After it's connected, run this on both servers:

```
SELECT * FROM ykval.yubikeys;
```

The table should be identical on both machines. It is updated every time you connect to VPN and use your OTP device.

Disconnect VPN. Edit the VPN client config to connect to the secondary server. Run that SELECT again on both servers. It should change again, and it should be identical on both servers.

VPN tunnel for MySQL replication

If the VPN servers are in different locations, and a secure connection does not exist already between the two, you must protect the MySQL replication channel with its own VPN tunnel. This should be easy, since we already have all the software we need.

Moreover, OpenVPN does not make a clear distinction between client and server. Sure, the server will listen for incoming connections, and the client will initiate, but beyond that they are almost the same. The same openvpn executable is used for both.

Finally, on most distributions (Ubuntu included), the openvpn service will look in the /etc/openvpn directory for *.conf files, and will happily create a new openvpn instance for each one of them. Just make sure to provision each *.conf file with the appropriate certificates and settings, and use different ports and/or different IPs for binding the servers to - so there’s no conflict between the multiple openvpn server processes.

You must setup separate openvpn processes just for replication, you cannot reuse the existing openvpn server processes, because the replication tunnel is different from all other tunnels, and different rules apply for firewall, routing, etc. The primary OpenVPN server will be a "server" for this tunnel, and the secondary will be a "client".

Create dedicated IP addresses for replication

Each MySQL instance needs to know the hostname or IP address of the other instance, for replication. You cannot replicate between the public IPs of those instances. You could use the private IP address for each instance, but in a dynamic environment (e.g. cloud) this may not work. Here's a way to make it work independently of the environment.

On each server, create an IP alias for the loopback interface, named lo:0, and give it a unique address (not in an IP range you're using somewhere else) and a netmask of 255.255.255.255. For example:

On both systems, edit /etc/network/interfaces and make sure this line exists at the end and is not commented out (create it if it's not there):

```
source /etc/network/interfaces.d/*.cfg
```

On the primary, create /etc/network/interfaces.d/000-lo0.cfg with the following content:

```
auto lo:0
iface lo:0 inet static
```

```
name lo Alias
address 10.97.83.1
netmask 255.255.255.255
```

On the secondary, create /etc/network/interfaces.d/000-l0.cfg with the following content:

```
auto lo:0
iface lo:0 inet static
name lo Alias
address 10.97.83.2
netmask 255.255.255.255
```

Reboot both systems. In some cases reboot might be a bit slow (couple minutes) after adding this IP alias to loopback - I don't know why. But the instance still works once it has rebooted.

Now if you do 'ifconfig' you should see this **on the primary**:

```
lo:0      Link encap:Local Loopback
          inet addr:10.97.83.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

And **on the secondary**:

```
lo:0      Link encap:Local Loopback
          inet addr:10.97.83.2  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

These interfaces keep their IP addresses no matter what happens to the public or private IPs of the instances. They are routable through VPN, and therefore could be used for replication.

Create VPN certificates for the replication tunnel

Go to your CA and create the server certificates for the primary server:

```
cd <secure-location>/easy-rsa
. vars
./build-key-server replication-srv001
# accept the defaults, sign and commit
```

Create the client certificates for the secondary server:

```
./build-key replication-cln001
```

```
# accept the defaults, sign and commit
```

Check the status:

```
# ls -l keys/replication-*
-rw-r--r-- 1 root root 5550 Sep 10 12:58 keys/replication-cln001.crt
-rw-r--r-- 1 root root 1098 Sep 10 12:58 keys/replication-cln001.csr
-rw------- 1 root root 1704 Sep 10 12:58 keys/replication-cln001.key
-rw-r--r-- 1 root root 5671 Sep 10 12:56 keys/replication-srv001.crt
-rw-r--r-- 1 root root 1098 Sep 10 12:56 keys/replication-srv001.csr
-rw------- 1 root root 1704 Sep 10 12:56 keys/replication-srv001.key
```

Configure the primary server for the replication tunnel

Copy the server certificates to /etc/openvpn:

```
cp keys/replication-srv001.crt keys/replication-srv001.key
/etc/openvpn/
```

Go into /etc/openvpn and generate a TLS auth key just for this tunnel:

```
cd /etc/openvpn
openvpn --genkey --secret ta-repl.key
```

Create replication-srv001.conf - the configuration file for the replication VPN server:

```
############################
port 12345
proto udp
float
persist-key
dev tun0
ca ca.crt
cert replication-srv001.crt # edit to match
key replication-srv001.key # edit to match
dh dh2048.pem
server 10.214.208.0 255.255.255.0 # edit to match
route 10.97.83.2 255.255.255.255 # remote replication IP
push "route 10.97.83.1 255.255.255.255" # local replication IP
ifconfig-pool-persist ipp-repl.txt
client-config-dir ccd
client-to-client
```

```
keepalive 10 120
tls-auth ta-repl.key 0
comp-lzo
user openvpn
group openvpn
persist-key
persist-tun
status openvpn-repl-status.log
verb 3
############################
```

Use a port different from the main VPN service, to prevent conflicts.

Create /etc/openvpn/ccd. Within that directory, create a file called /etc/openvpn/ccd/replication-cln001 (the name matches the common name for the secondary server connecting via this tunnel) with the following content:

```
iroute 10.97.83.2 255.255.255.255
```

Stop all openvpn instances:

```
service openvpn stop
```

And now start all openvpn instances:

```
service openvpn start
```

Or you could issue commands to each openvpn instance separately:

```
service <stop|start> [instance-name]
```

The name of the instance is the name of the .conf file in /etc/openvpn.

Configure the secondary server for the replication tunnel

In a different location, create a .zip bundle with all files needed for the secondary:

```
cd
mkdir second
cd second/
cp <secure-location>/easy-rsa/keys/replication-cln001.crt .
cp <secure-location>/easy-rsa/keys/replication-cln001.key .
cp /etc/openvpn/ca.crt .
cp /etc/openvpn/ta-repl.key .
cd ..
zip -r second.zip second/
```

Move the .zip bundle to the secondary and extract all files into /etc/openvpn.

Create the configuration file replication-cln001.conf with the content:

```
#############################
client
dev tun0
persist-tun
proto udp
remote 172.28.128.215 12345
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
cert replication-cln001.crt # edit to match
key replication-cln001.key # edit to match
ns-cert-type server
tls-auth ta-repl.key 1
comp-lzo
verb 3
#############################
```

The address in the 'remote' statement must match the address (or hostname) of the primary server, and the port must match the port you've used on the primary for the replication tunnel.

Restart openvpn on the secondary:

service openvpn restart

On the secondary, ping the replication IP of the primary:

```
ping 10.97.83.1
```

On the primary, ping the replication IP of the secondary:

```
ping 10.97.83.2
```

Configure MySQL to replicate through the tunnel

Assuming you have MySQL replication already working, these are the steps to convert it to replicating via the tunnel. If it's not working already, use the steps indicated in a previous chapter (Setup MySQL master/master replication) and simply use the replication IPs when you run CHANGE MASTER TO on both servers.

On both systems, stop the slave:

```
SLAVE STOP;
```

On the primary, query the master status:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------------+-----------------
-+
| File             | Position | Binlog_Do_DB       | Binlog_Ignore_DB
|
+------------------+----------+--------------------+-----------------
-+
| mysql-bin.000009 |      107 | radius,ykksm,ykval |
|
+------------------+----------+--------------------+-----------------
-+
1 row in set (0.00 sec)
```

On the secondary, query the master status:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------------+-----------------
-+
| File             | Position | Binlog_Do_DB       | Binlog_Ignore_DB
|
+------------------+----------+--------------------+-----------------
-+
| mysql-bin.000009 |      107 | radius,ykksm,ykval |
|
+------------------+----------+--------------------+-----------------
-+
1 row in set (0.00 sec)
```

On the primary, change the master to point at the secondary's replication IP, and use the log file name and log position shown by SHOW MASTER STATUS from the secondary:

```
mysql> CHANGE MASTER TO MASTER_HOST = '10.97.83.2', MASTER_USER =
'replicator', MASTER_PASSWORD = '<repl-pass>', MASTER_LOG_FILE =
'mysql-bin.000009', MASTER_LOG_POS = 107, MASTER_CONNECT_RETRY = 10;
Query OK, 0 rows affected (0.01 sec)
```

On the secondary, change the master to point at the primary's replication IP, and use the log file name and log position shown by SHOW MASTER STATUS from the primary:

```
mysql> CHANGE MASTER TO MASTER_HOST = '10.97.83.1', MASTER_USER =
'replicator', MASTER_PASSWORD = '<repl-pass>', MASTER_LOG_FILE =
'mysql-bin.000009', MASTER_LOG_POS = 107, MASTER_CONNECT_RETRY = 10;
Query OK, 0 rows affected (0.01 sec)
```

On both systems start the slave:

```
SLAVE START;
```

Now test replication just like in the previous chapter (create table, etc). You could run SHOW SLAVE STATUS on one system, and correlate it with SHOW MASTER STATUS on the other.

You could reboot both systems and test replication again, to make sure the configurations are persistent.

Security

It’s a vast topic, and all the usual recommendations apply: keep the systems up to date, don’t run unnecessary services, limit access to these systems, etc.

I will address in detail network security.

The OpenVPN servers must only offer one service to the Internet: the openvpn port - by default, udp/1194. Access to all other services must be restricted. You may allow ICMP traffic inbound if you feel that’s appropriate, for testing with ping and so on. SSH must only be accessible from locations authorized to ssh into the systems.

Begin by reading this introduction to iptables for Ubuntu:

https://help.ubuntu.com/community/IptablesHowTo

It is recommended to disable all iptables control mechanisms described in that document (if any are active), and only rely on iptables-persistent.

Install iptables-persistent:

```
apt-get install iptables-persistent
```

Save the current firewall configuration:

```
service iptables-persistent save
```

Check the configuration:

```
# cat /etc/iptables/rules.v4
# Generated by iptables-save v1.4.21 on Wed Sep 10 12:17:07 2014
*filter
:INPUT ACCEPT [533:30248]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [312:26216]
COMMIT
# Completed on Wed Sep 10 12:17:07 2014
```

The numbers within square brakets are irrelevant (just traffic counters). What matters is that there are no rules, and the policies are permissive. If not, flush the firewall and save again:

```
service iptables-persistent flush
service iptables-persistent save
```

Check /etc/iptables/rules.v4 again, it should now look as shown above.

Create a script called fw.sh with the following content:

```
##############################
#!/bin/bash

service iptables-persistent flush

# local traffic is always accepted
iptables -t filter -A INPUT -i lo -j ACCEPT
# stateful filtering for already established connections
iptables -t filter -A INPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT
# generic VPN
iptables -t filter -A INPUT -p udp -m state --state NEW -m udp
--dport 1194 -j ACCEPT
# replication VPN
iptables -t filter -A INPUT -p udp -m state --state NEW -m udp -s
172.28.128.216/32 --dport 12345 -j ACCEPT
# MySQL master/master
iptables -t filter -A INPUT -p tcp -m state --state NEW -m tcp -i
tun0 -d 10.97.83.1/32 --dport 3306 -j ACCEPT
# ssh
iptables -t filter -A INPUT -p tcp -m state --state NEW -m tcp
--dport 22 -j ACCEPT
# icmp
iptables -t filter -A INPUT -p icmp -m icmp --icmp-type any -j ACCEPT
# log - remove it after testing is done
iptables -t filter -A INPUT -j LOG
# and drop
iptables -t filter -A INPUT -j DROP

# NAT
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
##############################
```

What the script does:

- allows all local traffic over 127.0.0.1
- allows already established connections to come in (iptables is stateful)

- allows the world to access the main VPN port
- allows the other server to access the replication VPN tunnel (uses the public IP of the other server as a source)
- allows MySQL connections coming in via the replication VPN tunnel, with the destination being the local replication IP
- allows ssh connections (you should add here allowed source addresses via the -s option to further restrict access - wide-open ssh access is not healthy)
- allows incoming ICMP
- logs everything else, then drops it

At the end, if you need to NAT the VPN clients, use the final line - otherwise remove it.

Create a version of this script on each server, adjust the IP addresses, and run it. Test everything - VPN, replication, NAT, ssh access, etc. If everything works okay, then run **on both servers**:

```
service iptables-persistent save
```

and then delete the fw.sh script (it was just a temporary tool). Your firewall configuration will be stored in /etc/iptables.

When everything is pretty stable in production, feel free to edit /etc/iptables/rules.v4, delete the line with -j LOG, then restart the firewall:

```
service iptables-persistent restart
```

You only need logging for troubleshooting, so you could add that line back in when something seems wrong on the network side.