SKS Keyserver

The following is an incomplete guide to compiling, setting up and using SKS. Hopefully this is enough to get you started, in addition there is a wiki available, where in particular https://bitbucket.org/skskeyserver/sks-keyserver/wiki/Peering should help getting a working installation.

Prerequisites

There are a few prerequisites to building this code. You need:

- OCaml-4.0 or later. Get it from http://ocaml.org
- Berkeley DB version 4.6.* or later. You can find the appropriate versions at http://www.oracle.com/technetwork/database/berkeleydb/downloads/index.html
- GNU Make and a C compiler (e.g gcc)

Verifying the integrity of the download

Releases of SKS are signed using the SKS Keyserver Signing Key available on public keyservers with the KeyID

```
0x41259773973A612A
```

and has a fingerprint of

```
C90E F143 0B3A C0DF D00E 6EA5 4125 9773 973A 612A.
```

Using GnuPG, verification can be accomplished by, first, retrieving the signing key using

```
gpg --keyserver pool.sks-keyservers.net --recv-key 0x41259773973A612A
```

followed by verifying that you have the correct key

```
gpg --keyid-format long --fingerprint 0x41259773973A612A
```

should produce:

```
pub   4096R/41259773973A612A 2012-06-27
Key fingerprint = C90E F143 0B3A C0DF D00E 6EA5 4125 9773 973A 612A
```

A check should also be made that the key is signed by trustworthy other keys;

```
gpg --list-sigs 0x41259773973A612A
```

and the fingerprint should be verified through other trustworthy sources.

Once you are certain that you have the correct key downloaded, you can create a local signature, in order to remember that you have verified the key.

```
 gpg --lsign-key 0x41259773973A612A
```

Finally; verifying the downloaded file can be done using

```
gpg --keyid-format long --verify sks-x.y.z.tgz.asc
```

The resulting output should be similar to

```
gpg: Signature made Wed Jun 27 12:52:39 2012 CEST
gpg:                using RSA key 41259773973A612A
gpg: Good signature from "SKS Keyserver Signing Key"
```

Compilation and Installation

- Install OCaml and Berkeley DB
  When installing ocaml, make sure you do both the `make world` and the `make opt` steps before installing. The later makes sure you get the optimizing compilers. (do make opt.opt if you want faster compilation. You can then set the environment variables `OCAMLC`, `OCAMLOPT` and `CALMP4O` to `ocamlc.opt`, `ocamlopt.opt` and `camlp4o.opt` respectively.)
  If your vendor or porting project supplies prebuilt binaries and libraries for Berkeley DB, make sure to get the development package as you will need the correct version include files.
- Copy `Makefile.local.unused` to `Makefile.local`, and edit to match your installation. At minimum you need to specify correct verion for LIBDB
- Compile

```
make dep
make all
make all.bc # if you want the bytecode versions
make install # puts executables in $PREFIX/bin, as defined
             # in Makefile.local
```

  There are some other useful compilation targets, mostly useful for development.
  - `make doc`

    creates a doc directory with ocamldoc-generated documentation of the individual modules. These are mostly useful as documentation to the source code, not a user's guide.

Setup and Configuration

You need to set up a directory for the SKS installation. It will contain the database files along with configuration and log files.

Configuration options can be passed in on the command-line or put in the `sksconf` file in the SKS directory. the `-basedir` option specifies the SKS directory itself, which defaults to the current working directory.

Sksconf and commandline options

The format of the sksconf file is simply a bunch of lines of the form:

```
keyword: value
```

The `#` character is used for comments, and blank lines are ignored. The keywords are just the command-line flags, minus the initial `-`.

The one thing you probably want no matter what is a line that says

```
logfile: log
```

which ensures that sks will output messages to `recon.log` and `db.log` respectively.

Membership file

If you want your server to gossip with others, you will need a membership file which tells the `sks recon` who else to gossip with. The membership file should look something like:

```
epidemic.cs.cornell.edu 11370
athos.rutgers.edu 11370
...
```

This file should be called `membership`, and should be stored in the SKS directory. Note that in order for synchronization to work, both hosts have to have each other in their membership lists. Send mail to sks-devel@nongnu.org to get other SKS administrators to add you to their membership lists.

IMPORTANT NOTE: if you include the server itself in the membership file, you should make sure that you also specify the `hostname` option, and that the selected hostname is exactly the same string listed in the membership file. Otherwise, the `sks recon` will try to synchronize with itself and will deadlock.

Outgoing PKS synchronization: mailsync file

The mailsync file contains a list of email addresses of PKS keyservers. This file is important, because it ensures that keys submitted directly to an SKS keyserver are also forwarded to PKS keyservers.

IMPORTANT: don't add someone to your mailsync file without getting their permission first!

In order for outgoing email sync's to work, you need to specify a command to actually send the email out. The default is `sendmail -t -oi`, but you may need something different.

Incoming PKS synchronization

Incoming PKS synchronization is less critical than outgoing, since as long as some SKS server gets the new data, it will be distributed to all. Having more hosts receive the incoming PKS syncs does, however, increase the fault-tolerance of the connection between the two systems.

In order to get incoming mail working, you should pipe the appropriate incoming mail to the following command via procmail:

```
sks_add_mail sks_directory_name
```

Here's an example procmail entry:

```
PATH=/path/of/sks/exectuables

:0
* ^Subject: incremental
| sks_add_mail sks_directory_name
```

Built-in webserver

You can server up a simple index page directly from the port you're using for HKP. This is done by creating a subdirectory in your SKS directory called `web`. There, you can put an index file named `index.html`, `index.htm`, `index.xhtm`, or `index.xhtml`, supporting files with extensions .css, .es, or .js, and some image files with extensions jpg, jpeg, png or gif. Subdirectories will be ignored, as will filenames with anything other than alphanumeric characters and the '.' character. This is particularly useful if you want to run your webserver off of port 80. This can be done by using the -hkp_port command-line option.

Building up the databases

- First, you need to get a keydump. If you're running a PKS server, you should be able to convince PKS to generate one for you. If you're starting from scratch, you'll need to download one from the net. You should contact the pgp keyserver list pgp-keyserver-folk@flame.org
- in the SKS directory, put in a subdirectory called `dump` which contains the keydump files from which the database is to be built.
- Run sks_build.sh. That script actually runs three utilities. You might want to edit sks_build.sh if you want to trade off speed for space usage. At the current settings, you could run out of ram if you try this with less then 256 megs of RAM.

DO NOT DELETE THE `dump` DIRECTORY, even after the database is built. The original keys are not copied to the database, and so the dump must be left in place.

Platform specific issues

FreeBSD

On FreeBSD it appears that libdb is named differently than on some other platforms. For that reason, you need to set the LIBDB environment value to `-ldb46` instead of `-ldb-4.6` for other platfomrs.

Recent activity

Wiki

Clone wiki

sks-keyserver / Peering

View History

Getting started in the peering mesh, establishing peers.

Introduction

A keysever without keys is not very useful. SKS keyservers *gossip* to each other to exchange keys automatically and stay up-to-date. You should peer with a few other servers to join the mesh.

Details

You are assumed to have installed the SKS program and to have read the man page.

The steps are:

- make sure you have a new enough version of SKS
- double-check DNS
- set up reverse HTTP proxy on HKP port (highly recommended, best practice)
- get an initial keydump
- optionally, tune some keyserver configuration
- ask for peers, providing relevant information
- add the peers to your membership file
- endgame

There are two configuration files of interest here, both in your SKS basedir:

- `sksconf` : various configuration settings
- `membership` : your list of peers

The SKS basedir is either given on the command-line via `-basedir` or, more commonly, just the current working directory of the process.

Assuming your machine is `keyserver.example.com`, the URL you need to know about is: `http://keyserver.example.com:11371/pks/lookup?op=stats`

For advanced configuration, see also TLS Configuration

SKS versions

Short version: install the latest version available to you; when this text was last updated, that was 1.1.6

If your OS only packages older versions, please find a decent backports repository or install from source.

When this text was most recently updated, the minimum version to be listed in the main public pools is SKS 1.1.5; version 1.1.6 is currently required for the "subset" pool and is strongly recommended.

Versions prior to 1.1.2 have a severe interoperability bug (POST requests for exchanging keys are HTTP/0.9, does not work with modern setups having reverse HTTP proxies in front as a best practice).

Version 1.1.3 fixed lookups based on long-form keyids, so this is the minimum version for good compatibility with client software.

Version 1.1.4 fixed HTTP response codes to be friendlier to clients, fixed stability issues on Windows and VMs, and added support for elliptic-curve keys.

Version 1.1.5 stopped using short-form keyids for returning data.

Version 1.1.6 added support for keys based on Curve25519 cryptography.

Double-check DNS

Mistakes in DNS will be costly. It pays to review this, and perhaps tune your configuration ahead of time.

Each SKS server gossips to the other peers in its `membership` file. Each SKS server accepts incoming connections from IP addresses matching hosts in its `membership` file. Thus `membership` is both who you talk to and who you let talk to you.

You probably want a DNS hostname of `keyserver` or `sks` or `pgp-keys` in your domain. Eg, `keyserver.example.com`. You need that hostname to resolve to the IP on which your SKS server listens **and** which will be used for outbound TCP connections. You want to use a dedicated hostname for this, so that you can move the SKS service independently of other services (the SKS peering protocols do not support hacks like HTTP redirects).

If your machine has more than one IP address, it may be wise to set the `hkp_address` and `recon_address` options in your `sksconf` file. You should explicitly set all public addresses used, and if using a system with IPv4-IPv6 mapped addresses (in the form of ::ffff:192.168.0.1) avoid relying on the "::" catchall. Unless you are using a reverse proxy (see below), both options should be set to the same value(s). For example, assuming you have IPv6 connectivity and want to provide service on both IPv4 and IPv6:

```
hostname: keyserver.example.com

hkp_address: 192.0.2.42 2001:DB8::1:42

recon_address: 192.0.2.42 2001:DB8::1:42
```

(Strictly speaking: every address in `recon_address` needs to be in `hkp_address`, or covered by a wildcard address in `hkp_address`, or mapped back to an address in `hkp_address` via a reverse proxy, but they don't need to be the same. But part of using recon involves making connections to the hkp port on the same host.)

Note that `recon_address` serves two purposes: it defines which addresses your recon server will *listen* on, and defines preferred source IP addresses for *outbound* connections too. For instance, if you specify one IPv4 address in the option, then outbound IPv4 connections will use that source address, while outbound IPv6 connections will use the system default.

HTTP Performance

A more sophisticated and robust setup will place the HKP interface behind an HTTP "reverse" proxy which can receive complete requests and send complete responses, as SKS doesn't parallelise HTTP handling. It handles one request at a time, so slow network (deliberate or otherwise) on one TCP connection can DoS your server.

This is currently considered a **best practice**: you should do this; if you want to avoid doing this, you should take the time to understand the issues and make an informed decision to back that choice.

There is a minor, and disappearing, downside: the first release which correctly provided an HTTP version on the POST request was SKS 1.1.2; reverse proxies may legitimately drop the older

malformed requests (HTTP/0.9 and POST do not mix), so peers running releases older than 1.1.2 will fail to send you keys. Once no keyservers running at-least-1.1.2 are running without a proxy, there will effectively be a network split! Note though that the public DNS pools require a version newer than 1.1.2 (1.1.3 at last update of this text) anyway, so there will be no split in keyservers publicly listed in the pool addresses.

In addition, a reverse proxy adds complexity which can cause protocol weirdnesses that interact with existing deployed clients in unfortunate ways. Care **MUST** be taken in configuring these proxies to avoid breaking clients which are expected to still have broken versions in use for, likely, a decade or more.

A benefit is that most reverse proxies are extensively reviewed to be safe at serving files and may make a better choice than SKS for static file serving; by only passing the `/pks/` URL local paths through to SKS, you can reduce the attack surface of SKS and improve concurrency in file-serving.

To proceed, set the `hkp_address` to be localhost in your `sksconf` file:

```
hkp_address: 127.0.0.1 ::1
```

and run some kind of reverse proxy on the public IP address, passing the data on.

You **should not** put a proxy in front of port 11370, as the recon protocol is not HTTP. A peer can tie up your recon server (which is a reason to be judicious in your peering arrangements) but can't stop you serving keys.

Beware that for port 11371 traffic, you **must** be able to handle requests with *any* `Host:` header, for the various pools and CNAMEs which exist, and you **must** accept requests with no `User-Agent:` header set, as at least one major OpenPGP HKP client refuses to set a User-Agent field when talking to keyservers.

nginx

With nginx you might configure this (being sure to provide a correct hostname in the added `Via:` header:

```
server {
        listen  192.0.2.42:11371  default_server;
        listen  [2001:DB8::1:42]:11371  default_server;
        access_log  off;
        location / {
                proxy_pass         http://127.0.0.1:11371/;
                proxy_pass_header  Server;
                add_header         Via "1.1 keyserver.example.com:11371 (nginx)";
                proxy_ignore_client_abort on;
                client_max_body_size 8m;

                # Disallow Search Engines to index keyserver search results
                add_header         X-Robots-Tag 'noindex, nofollow' always;
        }
}
```

Important: The `proxy_ignore_client_abort` directive is needed to minimize interoperability problems with GnuPG when built with the curl-shim mock-cURL implementation of retrieving keys: it uses TCP half-closes (at time of writing) which plays badly with nginx (because nginx treats these as signs of a client abort). *Note: this issue can result in clients either not seeing keys, or seeing incomplete key data with very little warning that the results have been truncated.*

In addition, if you want to use a Kqueue-based nginx (BSD/MacOS system) and the SPDY patch for some web-sites, then you **must** use at least version **1.3.14 of nginx**, with at least version 66 of the SPDY patch. Prior versions effectively hard-coded the client-abort functionality early on, before the proxy dispatch, with no way to override this, and the change applying to all websites, not just those using SPDY.

More generally, of note is that the above preserves the conventional expected privacy of HKP requests by not logging details.

Also, bear in mind that a public keyserver will appear in DNS as part of public pools, so vhost constraints **must not** require a known Hostname: value to match on the 11371 port. With just one server specification matching that IP:port combination, it is also the default which will handle unrecognised hostnames. If your configuration style is to reject unknown hostnames by default with a dummy site, be sure to *not* do this for HKP. In the above, we're very explicit about this with the `default_server` parameter and by not setting a `server_name`.

Some unusually large keys are larger than 1MB, which is nginx's default value of `client_max_body_size` -- even 2MB is likely to be sufficient for the near future, but the above bumps the limit to 8MB. Tune according to taste, but running with the default may result in your keyserver being excluded from popular pools.

(nb: some older releases of nginx use `default` instead of `default_server`)

There is a more detailed example in TLS Configuration.

Apache

An Apache configuration is then:

```
## adjust these paths as appropriate for your OS/Apache combination:
LoadModule proxy_module libexec/apacheNN/mod_proxy.so
LoadModule proxy_http_module libexec/apacheNN/mod_proxy_http.so

Listen 192.0.2.42:11371
Listen [2001:DB8::1:42]:11371
## do *not* set NameVirtualHost on this host:port combination!
## For :11371, we use IP/port virtual-hosting, not names, accepting
## any pool name.

<VirtualHost *:11371>
    ServerName keyserver.example.com
    CustomLog /dev/null common
    <Proxy *>
```

```
            Order deny,allow
            Allow from all
    </Proxy>
    ProxyPass / http://127.0.0.1:11371/
    ProxyPassReverse / http://127.0.0.1:11371/
    ProxyVia On
    SetEnv proxy-nokeepalive 1
</VirtualHost>
```

If you must use `force-proxy-request-1.0` then be aware that this causes real-world interoperability breakage by default. When Apache knows that a backend is HTTP/1.0 and it receives an HTTP/1.1 request with `Expect: 100-continue` as a header, it rejects it with a 417 error (Expectation Failed). SKS is HTTP/1.0, yes, but the cURL project's libcurl sets that header by default, so this breaks retrieval by any client using libcurl, which includes normal builds of GnuPG. This applies from Apache 2.2.10 onwards in the 2.2.x release series, or any version of Apache 2.4.x. This does not affect Apache 1 or Apache 2.0.x.

So, if you want to downgrade this backend, then you **must also strip the Expect: header**, like so:

```
    SetEnv force-proxy-request-1.0  1
    RequestHeader unset Expect early
```

(If a future version of Apache implements the suggested logic of RFC2616 and learns which backends are HTTP/1.0, then you'll need to use this `RequestHeader unset Expect early` then too. For now, it's unneeded but doesn't hurt.)

Apache will use the only server available on a port, or the one with a `ServerName _default_`, or some other algorithm to match when there are multiple matches, so this vhost will still be used for other pool hostnames, which matches expectations for port 11371. The only reason for the `ServerAlias`directives is to make it easier to find the matching hostnames in your configuration.

If you do not configure a default `CustomLog` outside of vhosts, then you can better disable logging for this vhost by just not including a log directive for it. In the (common?) case of a default CustomLog, you need to override with a write to the bit-bucket `/dev/null`

The `ProxyVia` directive is needed so that pool-maintenance spiders can tell that there is a proxy in place and classify you accordingly.

Apache's `LimitRequestBody` value defaults to 0, for unlimited posts. If you have changed this default, ensure that the value is large enough to handle larger keys; see the nginx discussion of `client_max_body_size` for more details.

lighttpd

This example is for lighttpd 1.4.x. Apparently the proxy bits to come in 1.5 are much improved, but it has yet to be released.

Ensure these modules are enabled:

```
server.modules = (
# ...
    "mod_setenv",
```

```
    "mod_proxy",
# ...
)
```

Then a workaround for the libcurl/Expect issue:

```
server.reject-expect-100-with-417 = "disable"
```

For HKP on port 11371, something like this. You can mix and match IPv4/6 hosts (e.g. your IPv6 front can target www_sks on 127.0.0.1).

```
# for IPv4
$SERVER["socket"] == "192.0.2.42:11371" {
    proxy.server = ( "" => ( ( "host" => "127.0.0.1", "port" => 11371 )))
    setenv.add-response-header = ( "Via" => "1.1 keyserver.example.com:11371
(lighttpd)" )
    accesslog.filename = "" # logging disabled; you can specify the log file here if
you like
}

# for IPv6
$SERVER["socket"] == "[2001:DB8::1:42]:11371" {
    proxy.server = ( "" => ( ( "host" => "::1", "port" => 11371 )))
    setenv.add-response-header = ( "Via" => "1.1 keyserver.example.com:11371
(lighttpd)" )
    accesslog.filename = "" # logging disabled; you can specify the log file here if
you like
}
```

lighttpd uses the `server.max-request-size` option to control the maximum size of POST requests and the default limit is 2GB. If you have changed this default, ensure that the value is large enough to handle larger keys; see the nginx discussion of `client_max_body_size` for more details.

Initial keydump

The gossip protocol is for *reconciliation* and works when the set of differences between your keys and the keys of your peers is comparatively small. If you are missing too many keys, then the peer can refuse to provide you with keys (and the code gets less efficient too).

So you **must** start with a **recent** keydump. See KeydumpSources.

See http://www.keysigning.org/sks/ (currently: https://web.archive.org/web/20140803190834/http://www.keysigning.org/sks/) for instructions on getting and installing a keydump.

At time of writing, a keydump has more than 5.0 million keys. If you do not have this many keys loaded, something has gone wrong.

ToDo: We should have setup documentation in this wiki and link to that; the setup docs can reference the external sites. Rationale: a problem which plagues people installing SKS is the mess of

links to various sites, half of which are defunct or down at any given time and trying to pick through to find **stable** docs is unnecessarily hard. The keysigning.org docs are good, but will they be around in five years time? If people are reading this wiki page, this wiki is here, so we should be self-sufficient.

ToDo 2: The above was prophetic, we need something not stuck in the Wayback Machine as a link target.

Tune configuration

First, think about your IP addresses and hostname. See above.

Some of the default settings are designed for machines more resource constrained than anything sold as a new server-grade machine in the past several years. If you have the luxury of using a modern box, you should make a couple of changes.

SKS generates statistics once per day. By default, it does not generate these at start-up. Although this is a CPU-consuming task, it's only a few seconds on any modern box; if you start SKS late in the boot sequence, you should be okay. The advantage of turning this on at start-up is that when you ask for peers, your peers can look at your running stats. They can see that yes, you have loaded an initial keydump, you can read instructions and you're safe to peer with. You may even get more peers, faster, by doing this.

Also, you can look at your own stats and confirm that you have all the keys you expect to have.

There are two ways that PGP keys are spread around. The SKS network is now predominant, but there are also email-based networks. Many people choose to disable using the mailsync approach, so we'll show that here. The email-based network operators are generally open to new servers mailing updates to them, though.

```
initial_stat:

disable_mailsync:
```

Note that the boolean options are enabled by including them, with the trailing colon (":"), but with nothing after the colon. This is not a typo. If you do put something there, those elements will end up parsed as extra words on the command-line, leading to confusing errors for some of the DB maintenance tools.

Virtual Machine & Windows issues

The current version of SKS (at the time this section of text was last updated) is 1.1.4; if you have at least that version, then you can skip this bit.

Versions of SKS up to and including version 1.1.3 have a flaw, in that they use a current timestamp as a unique key for the Berkeley DB storage used for PTrees. If the clock resolution is too low, multiple entries occur with the same timestamp, updates are lost and the DB becomes corrupted.

The patch which changes this appears to have fixed all reported problems, both for VMs and for running SKS on Windows.

If you are unable to upgrade, then: if you are running Linux inside a VM, pass `clocksource=tsc` as part of the kernel command-line in Grub/Lilo/... to switch the VM's timer-system away from Jiffies towards the Time Stamp Counter of the processors. Note that this can itself be problematic with

older kernels locking up on SMP instances. If running SKS in a VM instance, you should probably constrain it to a single CPU.

Ask for peers

Join the SKS mailing-list. It's called `sks-devel` but in practice it's also an operations list.

The list's sign-up page is at: http://lists.nongnu.org/mailman/listinfo/sks-devel

Send an email announcing your server and asking for peers.

When you send the mail, be sure to, at the very least, mention your own hostname!

If you PGP-sign the mail, then potential peers can see (1) that you can use PGP, (2) a keyid being used, which they can import into their own keyrings and perhaps use when talking to you. It's a mailing-list for a PGP keyserver, PGP signatures on mails are perfectly acceptable etiquette.

Ideally, you'll provide the template `membership` file line, including a contact address and keyid to use. When you add another peer to your own configuration file, you'll want some contact information so that when you debug problems, you know who to talk to. So provide that information yourself.

Some people have policies where they prefer to peer with "local" servers, because Internet traffic is cheaper and it provides more local resilience. So it's good to mention which country/region you're in.

Here's a rough template email, containing some bogus information but letting you pick what to supply. You can supply less information if you're not comfortable sharing. (But perhaps some people then won't be comfortable peering).

```
Subject: seeking peers for keyserver.example.com

Hi,


I am looking for peers for a new SKS keyserver installation.


I am running SKS version A.B.C, on keyserver.example.com.

{We are an ISP/university|this is a private machine|whatever}.

The server is physically located in Foobarlandy (EU).

The machine has IPv6 connectivity.


I have loaded a keydump from Foo, dated 2009-02-01.

I see NNNNNNN keys loaded.


For operational issues, please contact me directly.


keyserver.example.com 11370 # Fred Bloggs <fred@example.com>
0xDEADBEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEF
```

```
Thank you,

-Fred Bloggs
```

Add peers

Over the next couple of weeks, you'll get responses come in as people respond. Typically, the response will say that they have added you and will provide their contact details, in a line suitable for inclusion in your own `membership` file.

Add the line to your `membership` file. Wait for this to be picked up by your `recon` process. If this is your first peer and if you put `initial_stat:` into your `sksconf` file, you might restart SKS now; this will let you see in your stats page that the peer is present, that you did things correctly. Note that the stats page comes from the `db` process, the peering is managed by the `recon` process.

Look at the `*.log` files in your basedir. In particular, `db.log` and `recon.log` are likely to exist and be useful in tracking what's happening.

Note that the `membership` lines only provide the SKS recon port; key retrieval will happen on a port number one greater than the recon port. Thus recon lines are normally on port 11370 and retrieval happens on the normal HKP 11371 port.

The long haul

Activity on the `sks-devel` mailing-list can be sporadic. A month can go by with no posts, or there can be a several posts a day for a week. There is a very high signal-to-noise ratio. Loosely speaking, if you're running an SKS keyserver then you should stay subscribed. Reasons include:

- announcements of new releases
- responding to feedback from people proposing to change the software you run
- finding out about interoperability issues or SKS bugs
- finding out about new peers

New peers will come from other new entrants asking for new peers. Sometimes, they'll come from people who don't immediately add anyone who appears asking for a peer but if you post a couple of times, demonstrating competence, they'll be more confident in you and will ask to peer.

You should look at:

- http://www.sks-keyservers.net/status/ to see how your keyserver is being reported and check that everything is okay. If the sks-keyservers.net site reports that you are okay, then your keyserver becomes a candidate for automatic inclusion in the `pool.sks-keyservers.net` DNS pool.

You should think about log rotation and periodic database dumps.

Editorial bias note

This wiki page was written by someone who used to peer with almost anyone, but who really is happy when he sees a keyserver showing that it has a full keydump, as he's been bitten a couple of times by this issue.

Updated 2018-05-23