

# *SSL Delivery System*

Poulhiès Marc

`marc.poulhies@cern.ch`

CERN/EPFL

# Summary

---

- Current system
  - How it works
  - Security issues
- Proposal for next system
  - Requirements
  - How it works
  - Security improvements
  - Weak points

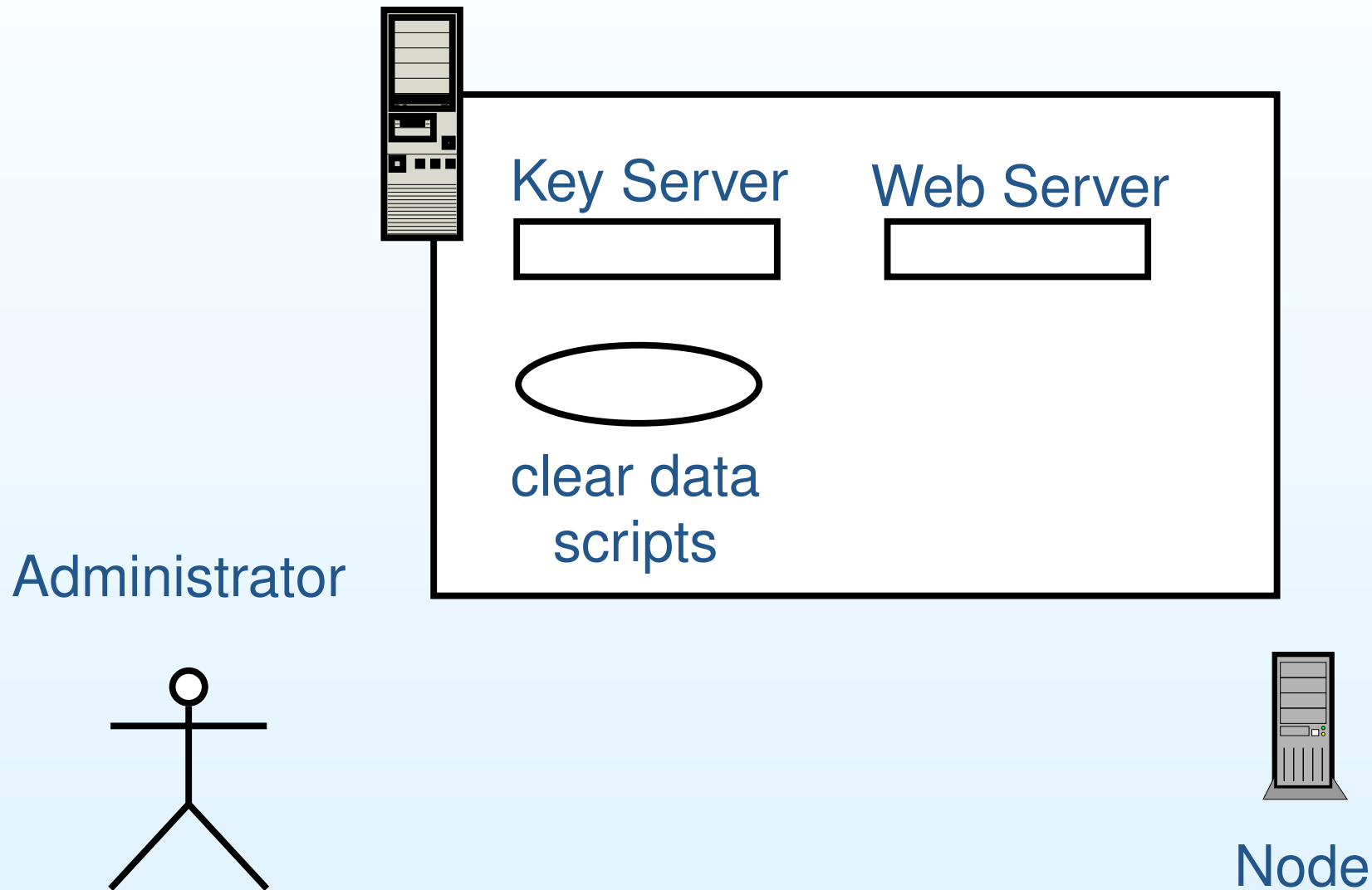
## Current System

Uses standards/well known things:

- SSH/SCP when secure connections are needed
- GPG for data encryption
- HTTP for data transport when secure connections are not needed

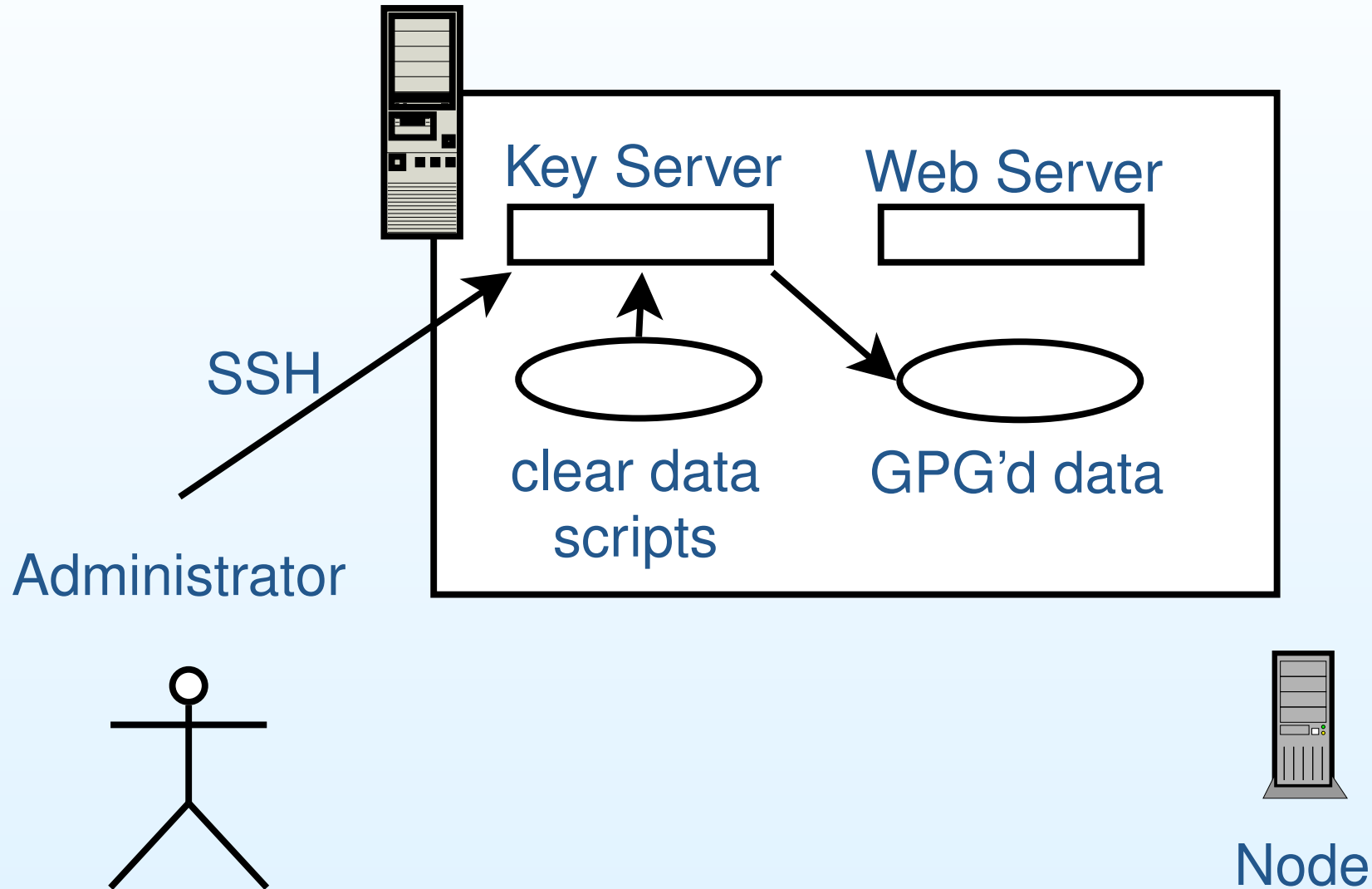
# How it works

“presentations”



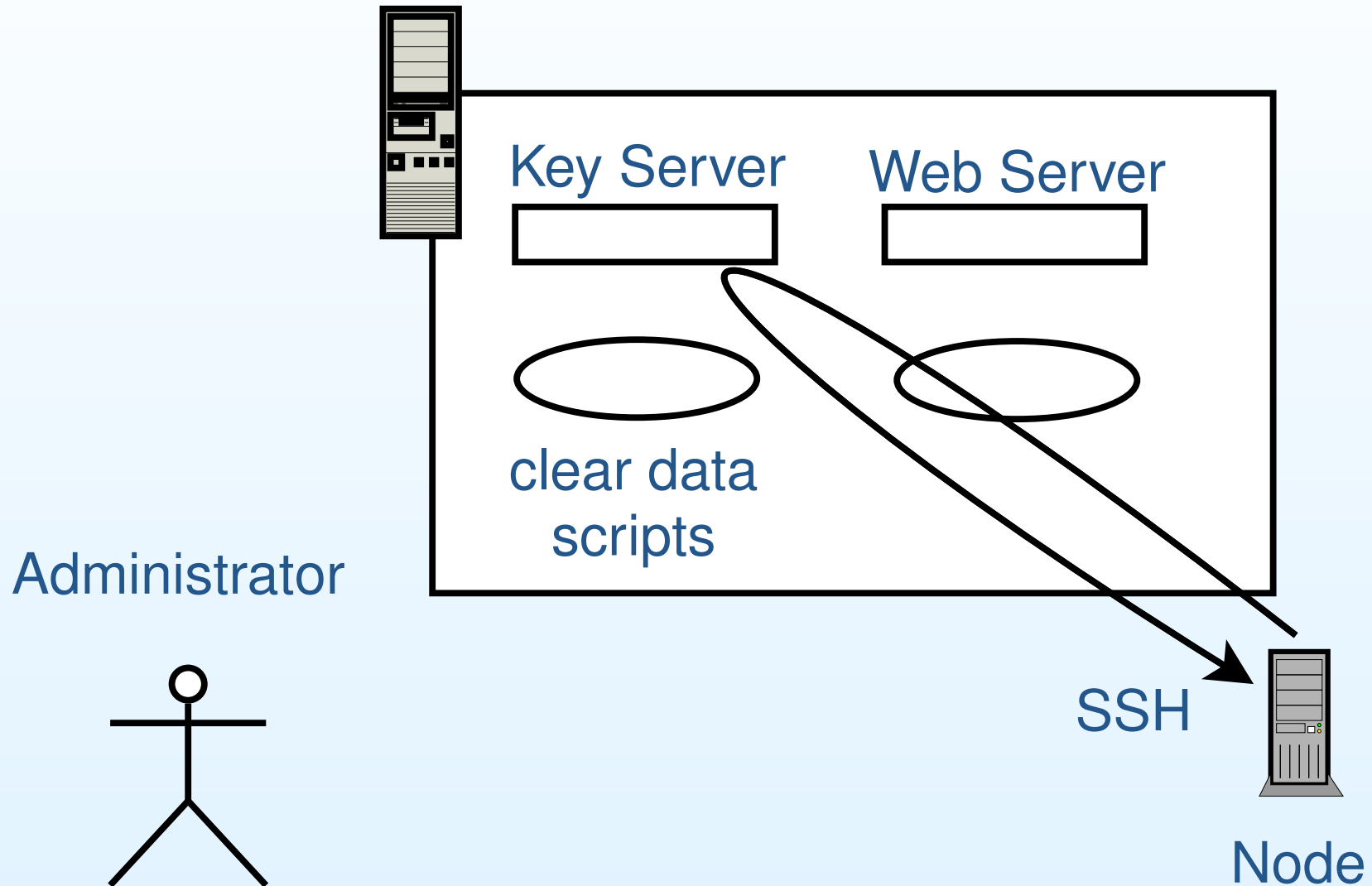
# How it works

Admin “prepares the machine”



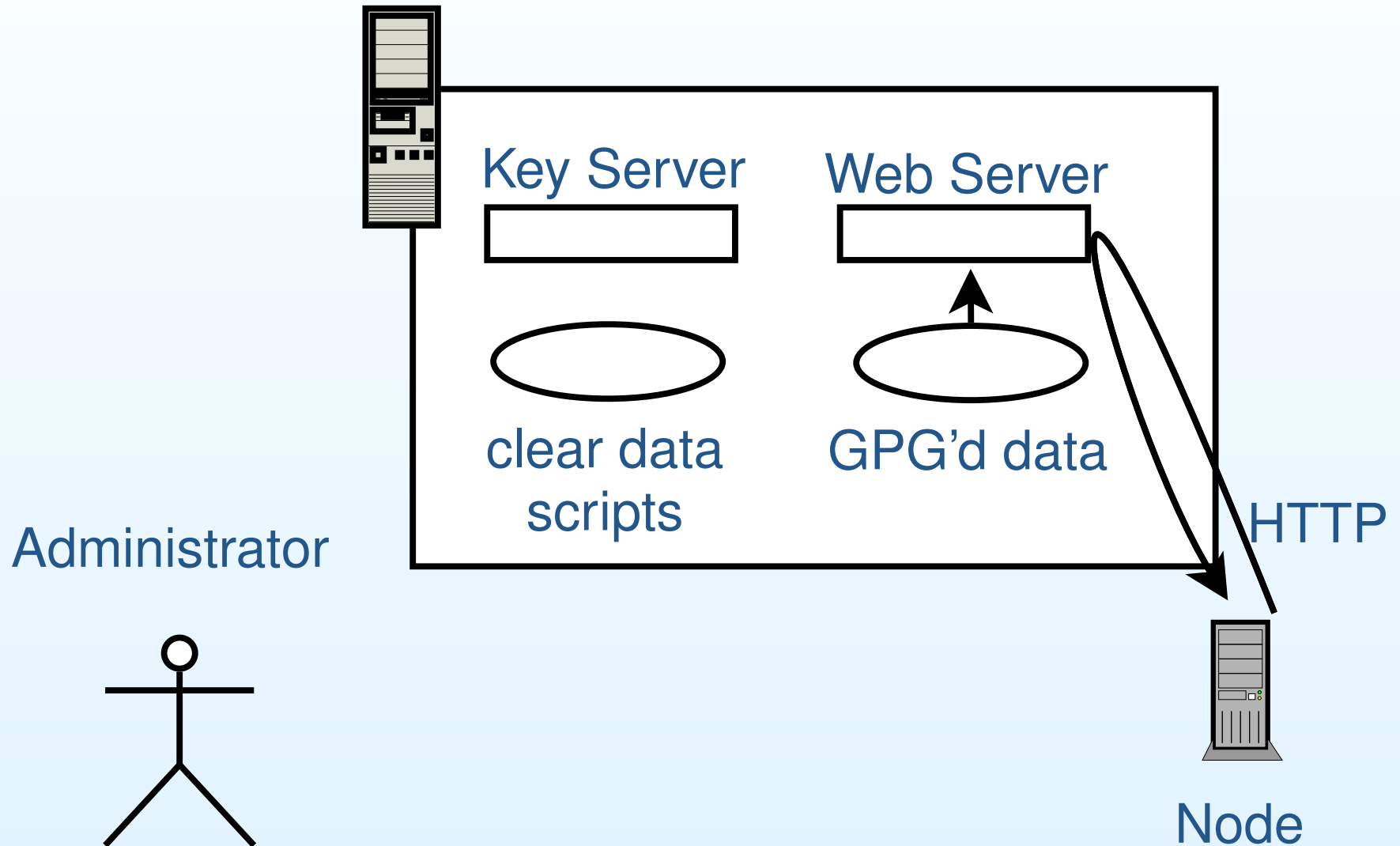
# How it works

Node gets its GPG key-pair



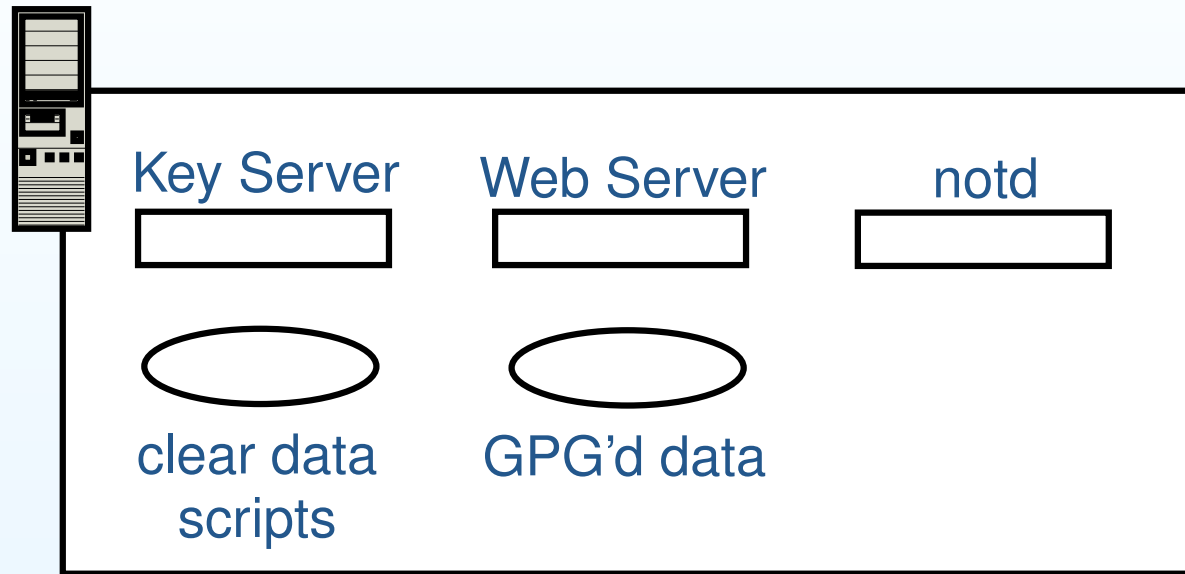
# How it works

Node gets some secured files

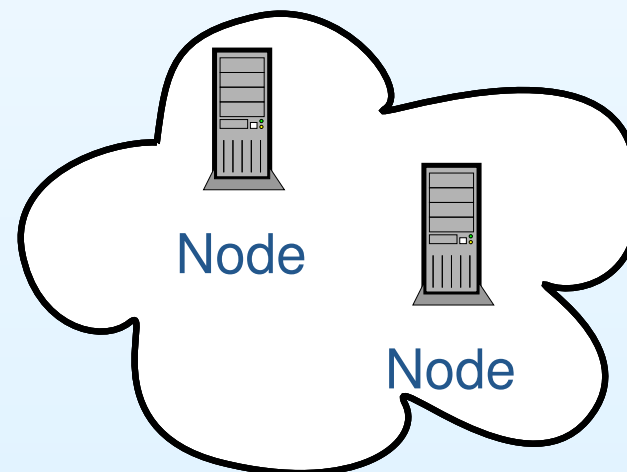
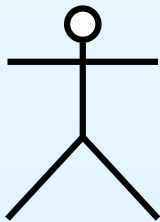


# How it works for updates

“presentation”



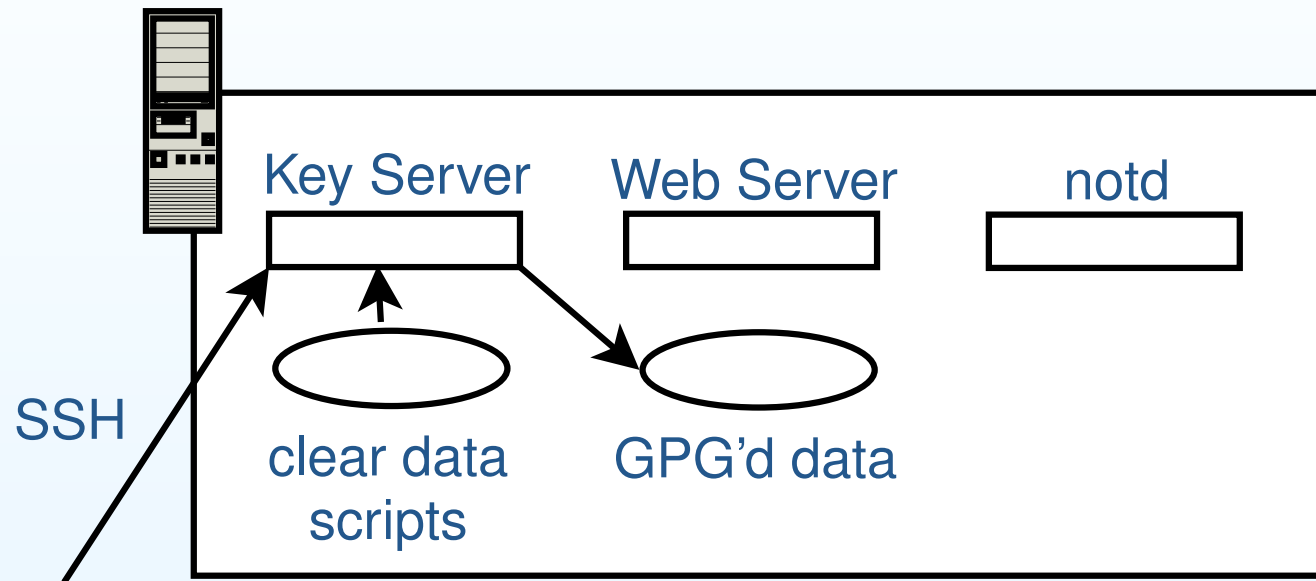
Administrator



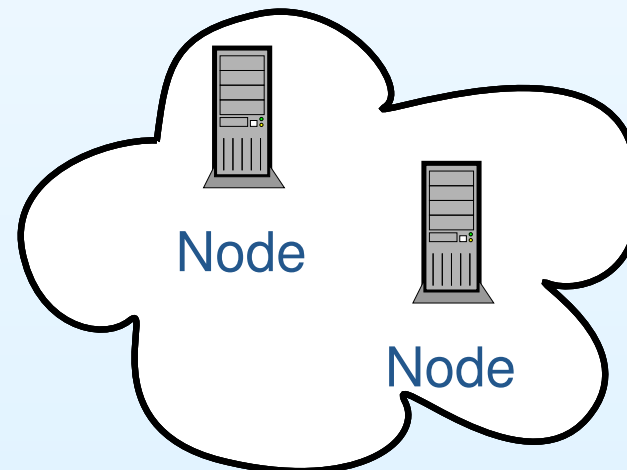
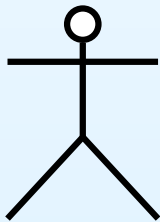


# How it works for updates

The admin updates files

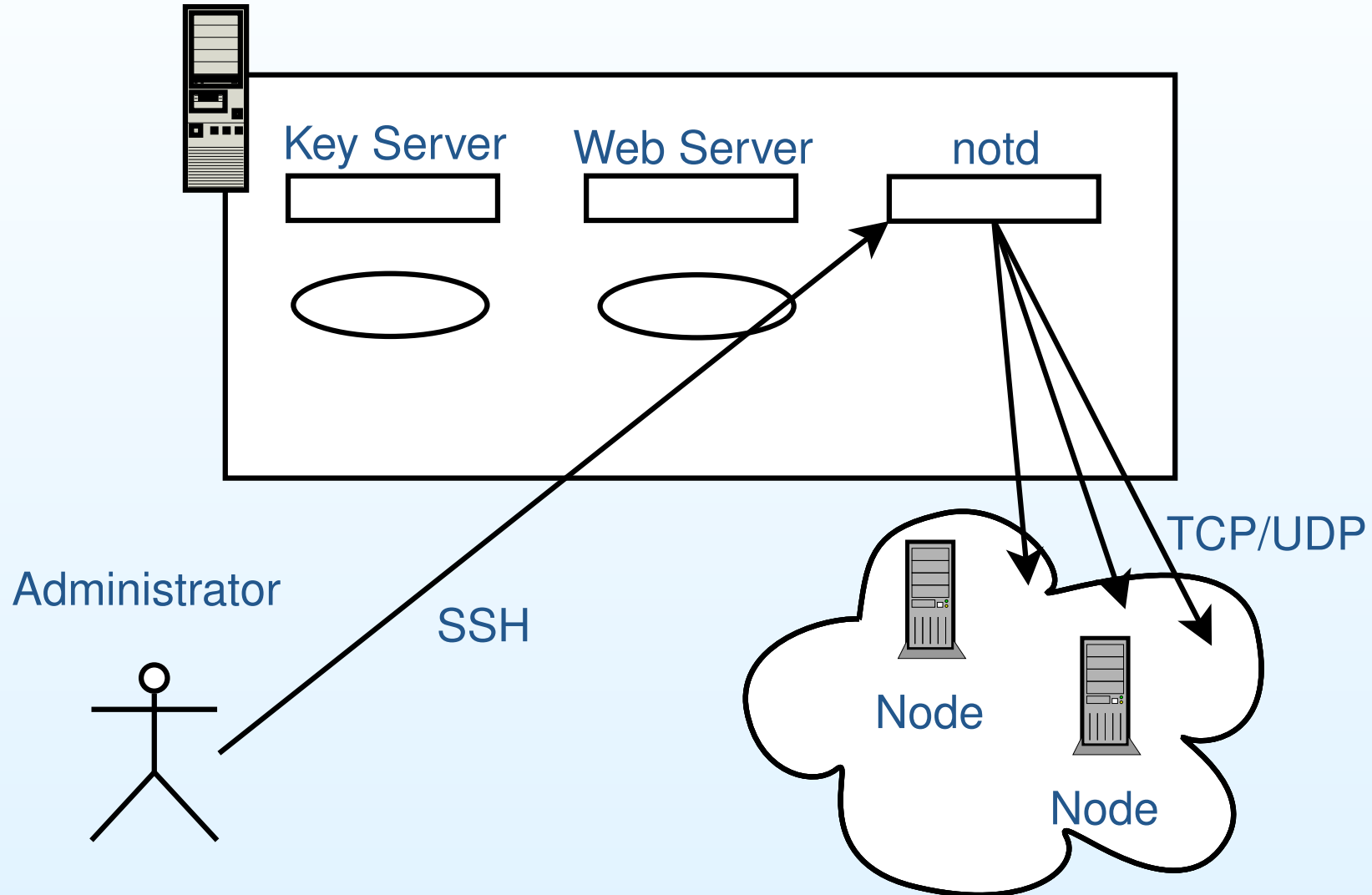


Administrator



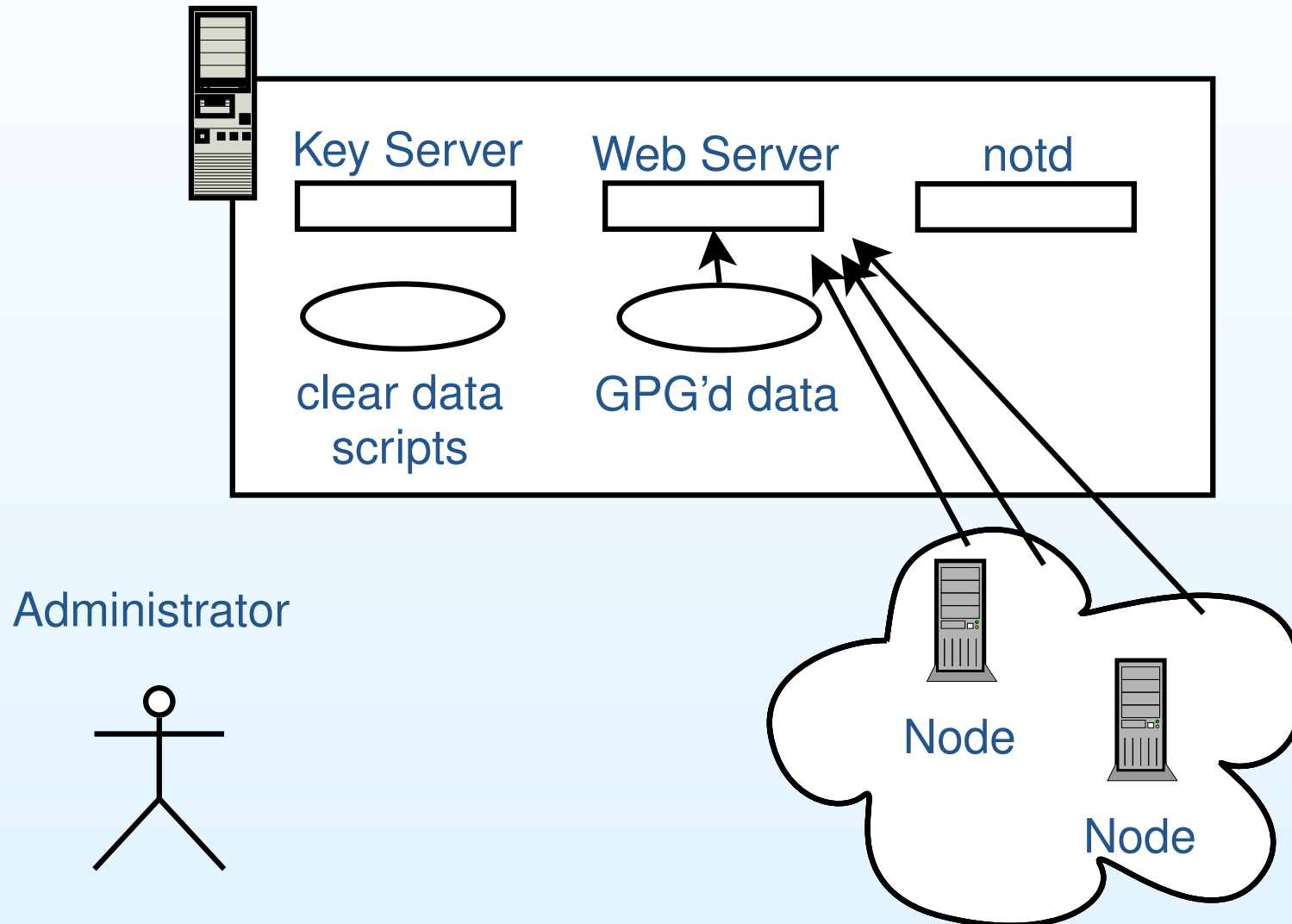
# How it works for updates

The “notd” notifies nodes



# How it works for updates

Nodes get their files



## Security issues

System suffers from different security problems:

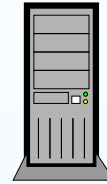
- IP/MAC as identifier
- No control on downloads
- Private part of GPG keypair duplicated
- No authentication on data

# IP as identifier

---

Presentation

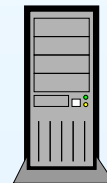
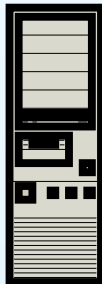
Web Server



Bob

192.168.0.1

Key Server



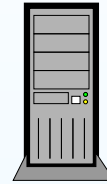
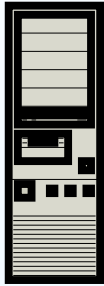
Eve

192.168.0.2

# IP as identifier

Bob gets its GPG KP

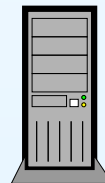
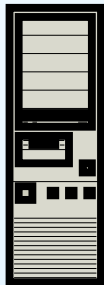
Web Server



Bob

192.168.0.1

Key Server



Eve

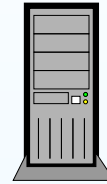
192.168.0.2



## IP as identifier

Eve spoofs Bob's IP and gets Bob's GPG KP

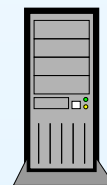
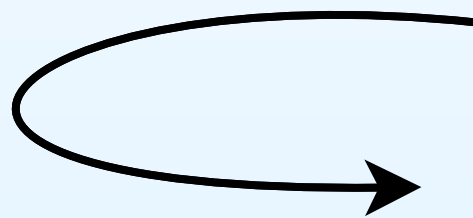
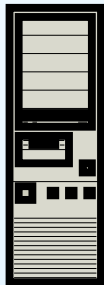
Web Server



Bob

192.168.0.1

Key Server



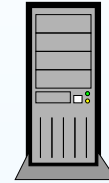
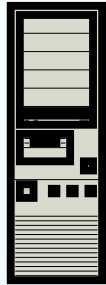
Eve

192.168.0.1/2

## IP as identifier

Both Bob and Eve can download files for Bob

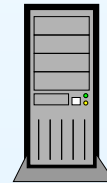
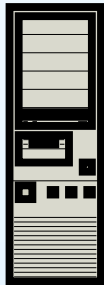
Web Server



Bob

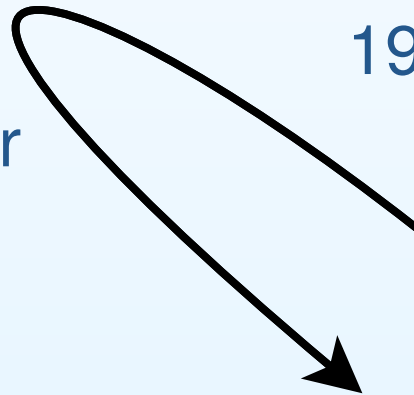
192.168.0.1

Key Server



Eve

192.168.0.2





## Other problems

---

- If a machine can download its GPG key, it can also download all other's keys and therefore download/read all files on the data repository.
- GPG keys are duplicated (on server and on client). This was a previous requirement but is not needed anymore.
- Node getting a secured file can verify this file was really for him as he can read it using its private key, but he can't verify who made this file.

## New system

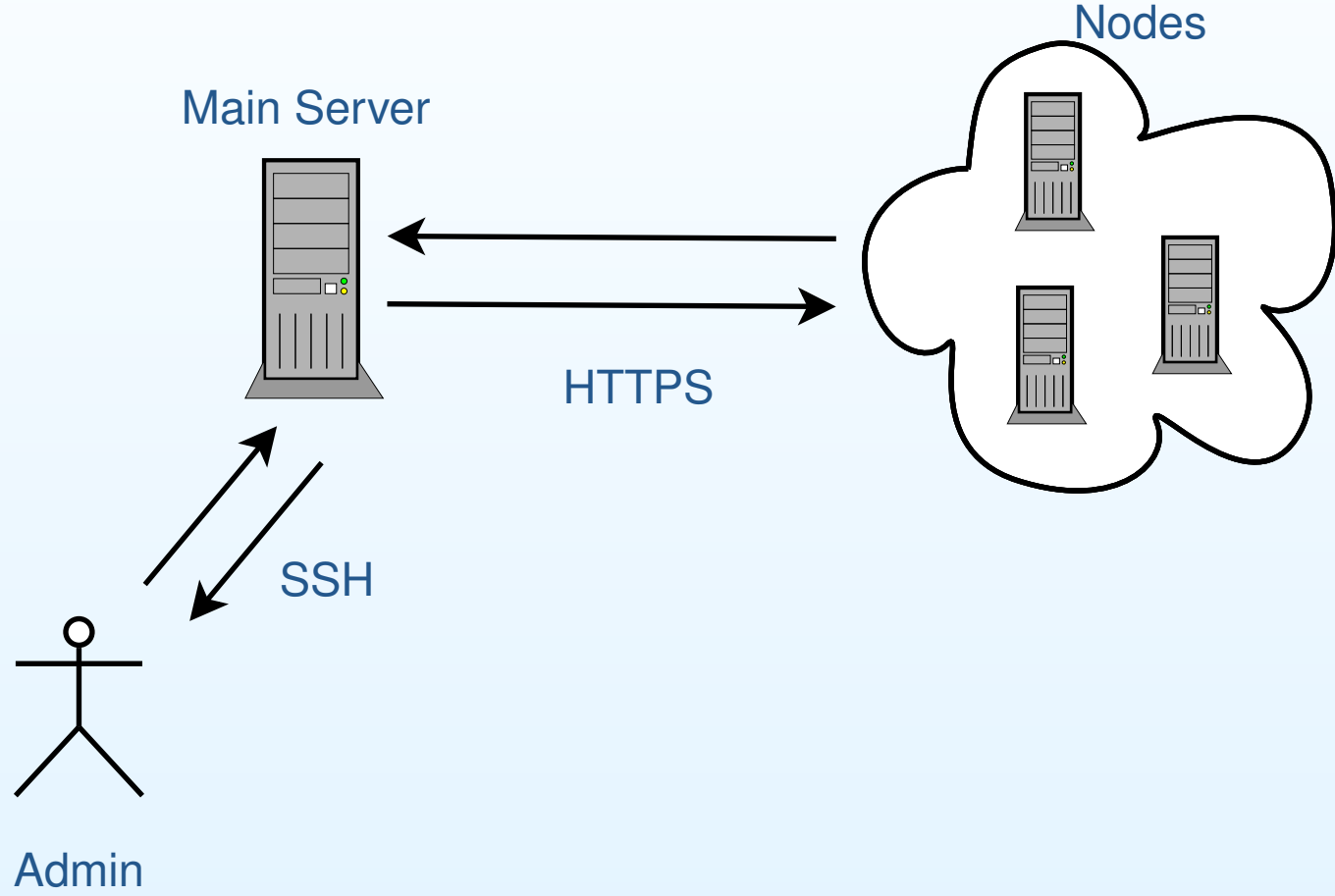
The new system must correct or propose workarounds for previous problems.

After a talk with security team, it was decided that it will be kept as simple as possible:

- a “unique” computer for server/data repository.
- HTTPS:
  - for secure data transport
  - for client/server authentication
- SSH for interactive connections (administration interface)

# Overview

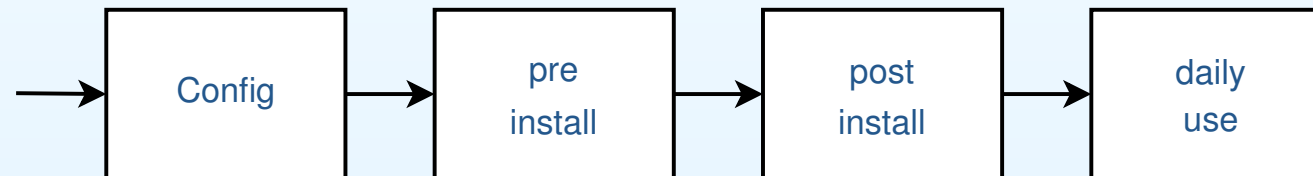
Quick overview of the new system:



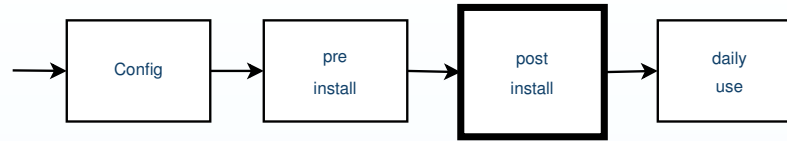
## Overview (cont.)

There are different parts in the system:

- the configuration (CDB)
- the preparation (pre-install of the nodes)
- the node initialization (post-install of the nodes)
- the daily use

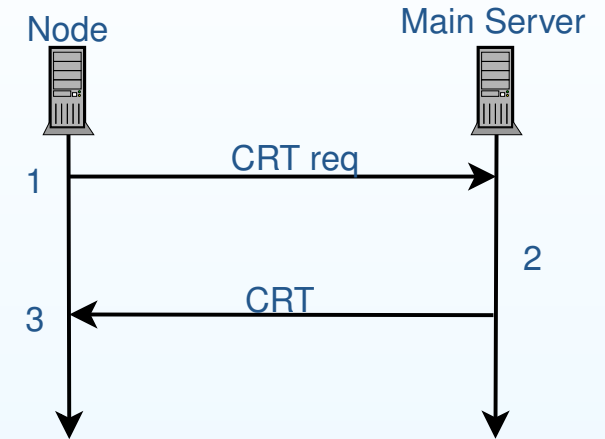


## post-install

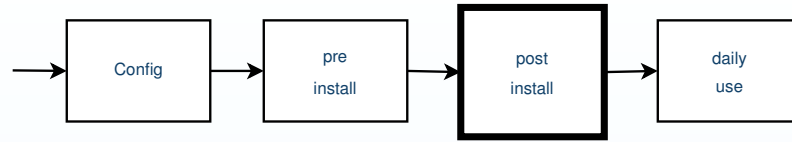


1. key generation + certificate request creation + server authentication
2. client authentication (weak) + check node is allowed + reply with a signed certificate
3. The node is now able to authenticate itself with its certificate.

First client authentication is weak because the server can only check node's identity with its IP. The checks in step 2 include the time-window mechanism.



## post-install

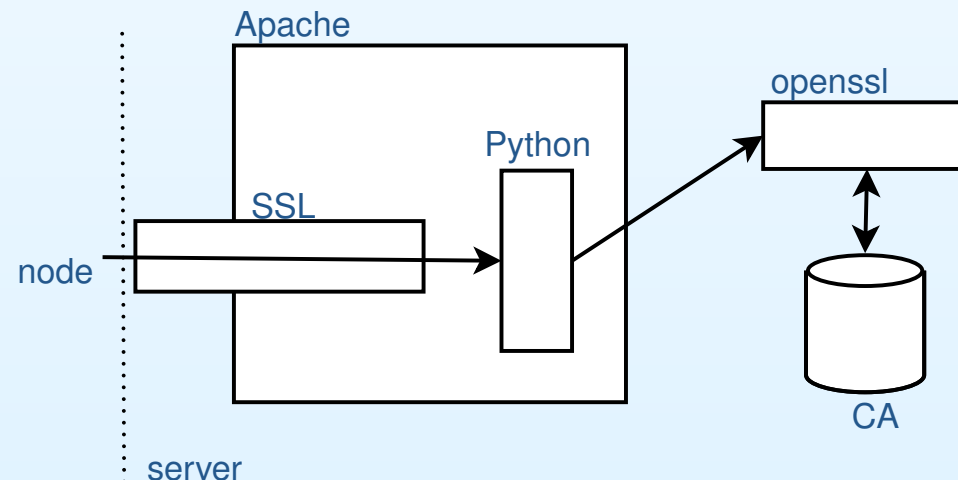


### Client side uses:

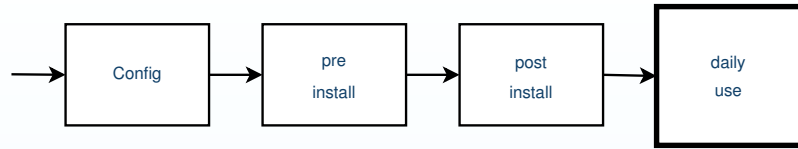
- curl command line for HTTPS communications with server.
- openssl command line for certificate handling.

### Server side uses:

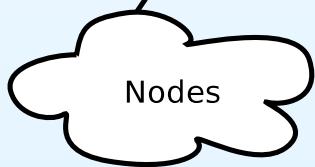
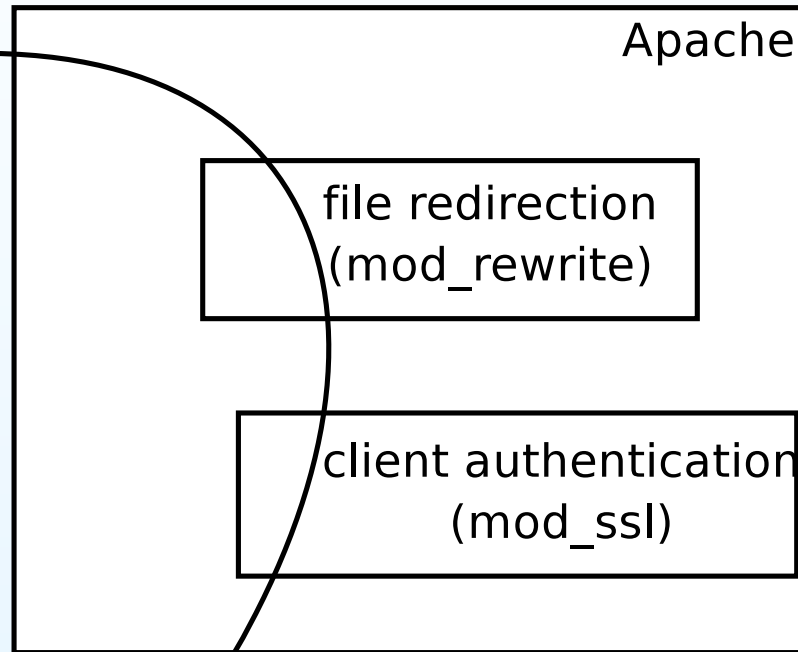
- Apache 2 with mod\_ssl for HTTPS.
- mod\_python and openssl command line for the certificate authority and various checks (ACL, DNS lookups, ...)



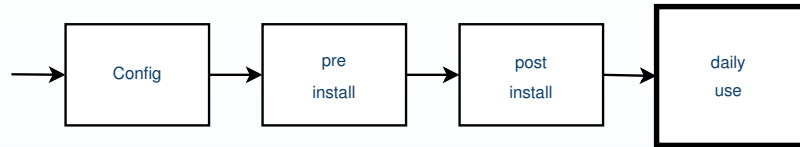
# daily use



```
-- lxplus  
| -- item1.tar.gz  
| -- lxplus056.cern.ch  
| | -- item1.tar.gz  
| | -- passwd.tar.gz  
| -- passwd.tar.gz
```



## daily use



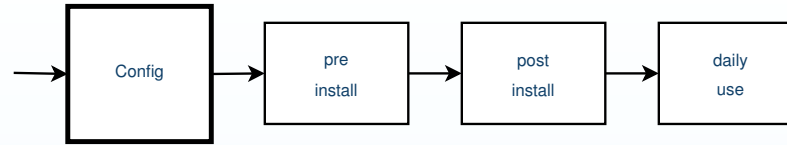
- node connects to server
  - client verifies server's certificate
  - server verifies client's certificate
- node requests a file with a URL:  
`/files/sensitive.tar.gz`
  - server checks for a node specific `sensitive.tar.gz`, if not
  - server checks for a cluster specific `sensitive.tar.gz`, if not
  - server checks for a global `sensitive.tar.gz`

All these operations are done with `mod_ssl` and `mod_rewrite`, so they should not be a big overhead.



## Node Configuration

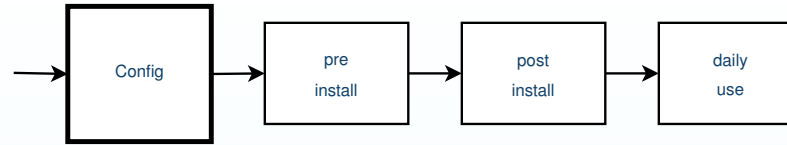
---



Define each item to be securely retrieved (to be integrated in CDB profile):

- a unique name (*passwd, ssh\_keys, grid\_cert, ...*)
- a method: *script* or *file*
- a level: *node, cluster* or *global*

## Node Configuration

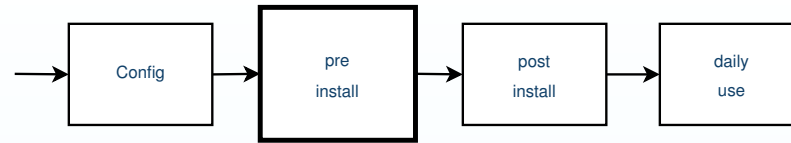


Here's a possible informations schema (this XML view is just an example) :

```
<nlist name="item1">  
  <string name="method">script</string>  
  <string name="level">cluster</string>  
</nlist>
```

- The **script** method is used when the clear data are to be generated; for example it may be possible to store clear passwords in some file and then, `crypt`(or hash,...) these to create a usable passwd file.
- The **cluster** level is used when you don't want to use the possible node specific files for this item.

## pre-install



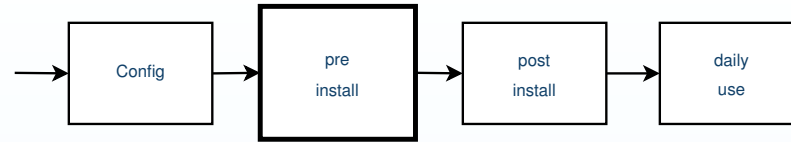
This must be done before a node gets installed. The basic idea is :

- admin designates which nodes will be installed
- system asks the admin to give some input (scripts, files)

System can find the scripts/files location based on informations from the node's CDB profile.

System will also take care of time-window and/or ACL for certificate requests.

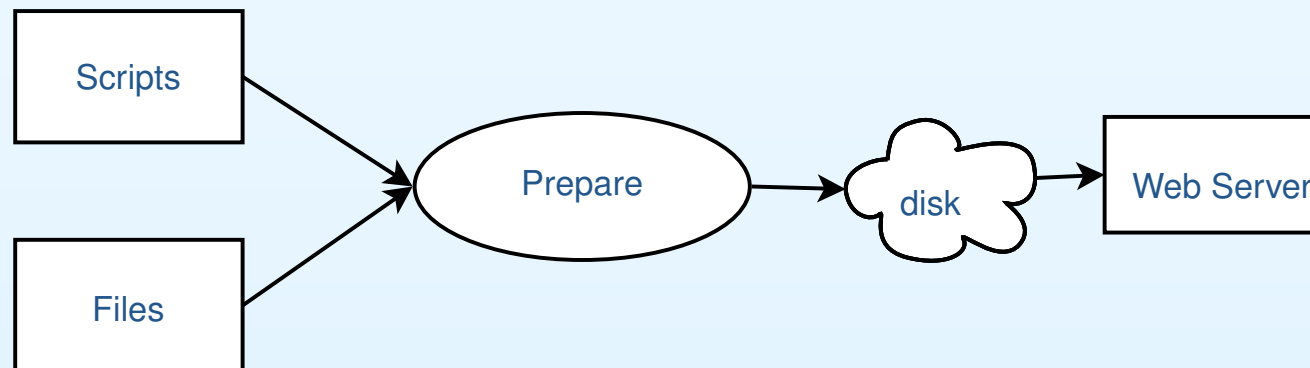
## pre-install



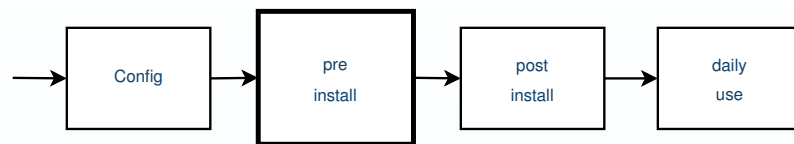
Before the admin can start the pre-install script, he must put the script for **item1** on the server. As we are dealing with cluster wide information, the script will be in

`$SCRIPT_ROOT/cluster1/default/item1/`.

It will have its own directory. It must be called `run` and will be invoked with some parameters (node's hostname, cluster's name, level and the output path).



## pre-install



Only then, the admin should try to run the pre-install process (custom shell):

```
[hostname?]> node_hostname
Method type for item1 is script, scope: cluster
-> got ./scripts/cluster1/default/item1/
Updating info for /cluster1/item1.tar.gz
-> OK
Found 1 item(s)
```

The script will be invoked with parameters:

```
-l <level> -h <hostname> -d <output>
```

Only the output is mandatory. It will be a file path pointing inside a different tree than the scripts' tree.

When method is **file**, the item file is to be copied to the correct directory.

## script tree

```
.
|-- cluster1
|   |-- default
|   |   |-- item1
|   |   |   '-- run
|   |   '-- item2
|   |       '-- run
|   |-- host1
|   |   '-- item1
|   |       '-- run
|   '-- host2
|       '-- item2
|           '-- run
'-- cluster2
...
```

## files tree

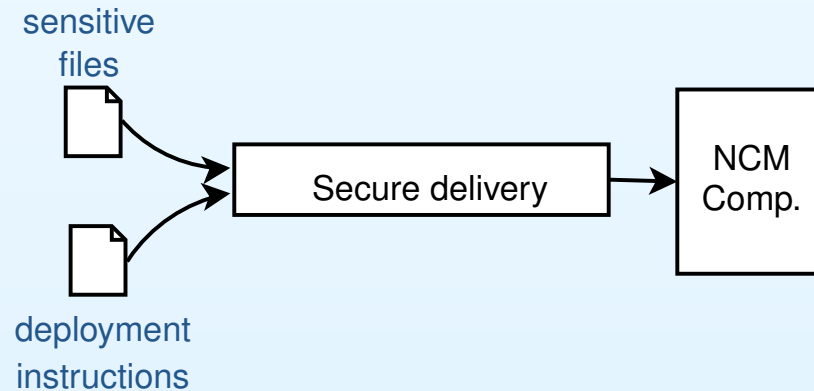
---

```
.
|-- cluster1
|   |-- host1
|       |-- item1.tar.gz
|   |-- host2
|       |-- item2.tar.gz
|   |-- item1.tar.gz
|   |-- item2.tar.gz
'-- cluster2
...
```

## Quattor (NCM)

One goal of the new system is to ease the addition of new file (don't want to modify the system for each file). One way for doing so would be to embed deployment instructions with the files themselves (in a way similar to .deb/.rpm packages). The instructions should be a shell script doing as less as possible.

The software part that will take care of it will be a NCM component.



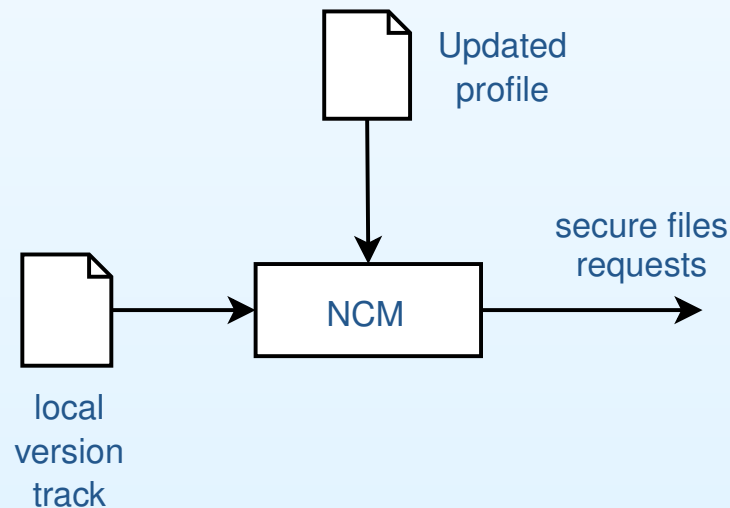


## Versioning & updates

Every *item* has a version number so that the software (NCM component) knows if it needs to request the item or not (on updates for example).

It must keep current installed versions. This can be done in a text file:

```
item = version
```



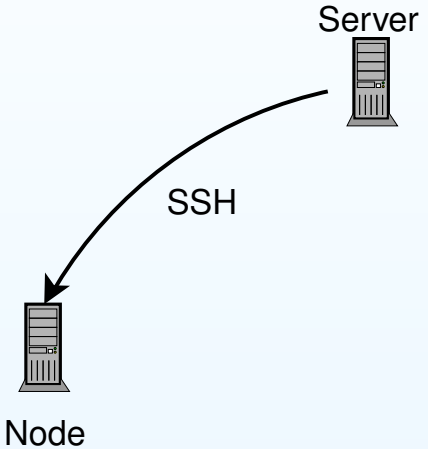
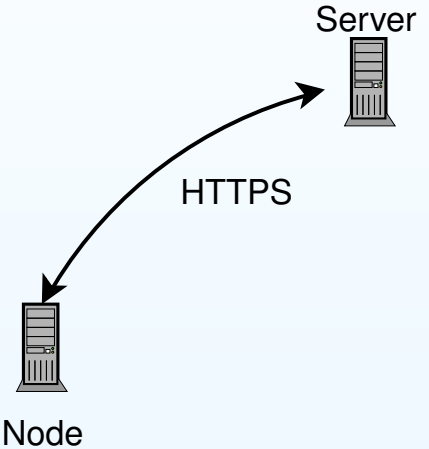
## Download count & time window

---

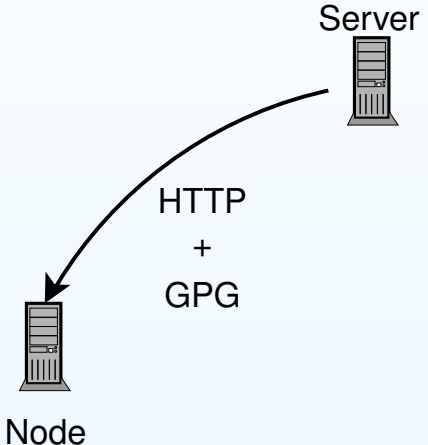
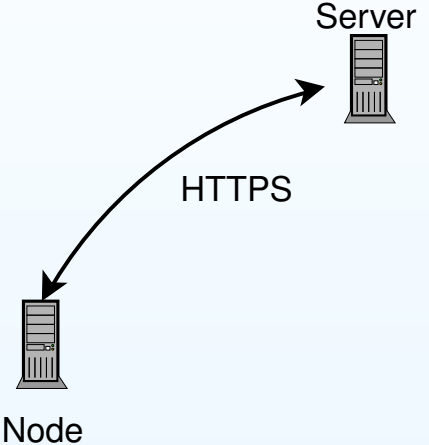
As the first node authentication is weak (only based on IP address), some actions are taken to minimise this weakness:

- for a given host, only one certificate can be issued. If there are some other requests for the same host, first certificate will be revoked and a warning transmitted to admins.
- certificate requests can only be sent inside a time window. Requesting a certificate outside this time window will be forbidden and a warning is also transmitted to admins.

## comparison (node bootstrapping)

Old	New
 <p>The diagram shows a 'Node' (represented by a server rack icon) on the left and a 'Server' (represented by a server rack icon) on the right. A curved arrow points from the Node to the Server, with the label 'SSH' positioned above the arrow.</p>	 <p>The diagram shows a 'Node' (represented by a server rack icon) on the left and a 'Server' (represented by a server rack icon) on the right. A curved arrow points from the Node to the Server, with the label 'HTTPS' positioned above the arrow.</p>
<ul style="list-style-type: none"><li>- weak client auth</li><li>- no server auth</li><li>- keys duplication</li></ul>	<ul style="list-style-type: none"><li>- weak client auth</li><li>+ server auth</li><li>+ no private data on network</li></ul>

## comparison (daily use)

Old	New
 <p>The diagram shows a Node (server icon) on the left and a Server (server icon) on the right. A curved arrow points from the Node to the Server. The text 'HTTP + GPG' is written in the middle of the arrow.</p>	 <p>The diagram shows a Node (server icon) on the left and a Server (server icon) on the right. A curved arrow points from the Node to the Server. The text 'HTTPS' is written in the middle of the arrow.</p>
<ul style="list-style-type: none"><li>+ recipient unicity</li><li>- no client authentication</li><li>- no issuer authentication</li></ul>	<ul style="list-style-type: none"><li>+ client auth</li><li>+ server auth</li></ul>

## comparison (cont)

Other differences of interest:

- New system won't duplicate information. If you have the same file for all cluster's node, there will be one file on the server. This was not true with previous system as every file was GPG'ed for every hosts.

## Known weaknesses

---

- The biggest one: if the main server is compromised, all sensitive informations stored are compromised.
- Scale:
  - tested with 50 nodes
  - more tests needed

This is the end!

Questions?

## Prototype example

- Apache configuration
- Server scripts
- Client scripts



## Apache conf (SSL)

Only one server, listening to SSL requests only. Two main directories:

- `CA/`: for requesting a certificate. No client authentication required.
- `rfiles/`: directories with sensitive files. Client authentication required.

In both case, client must check server's identity.

## Apache conf (SSL) (cont)

### Python handling for certificate requests:

```
<Directory /var/www/html/https/CA>
    AddHandler python-program .py
    PythonHandler mod_python.publisher
    PythonDebug On
</Directory>
```

### And the python script:

```
def send(req, csr):
    ...
    csr.write(csr.value)
    ...
    os.popen(openssl ca -batch -config ca.conf ...)
    ...
    return crt.read()
```

**Request via:** `https://serv/CA/get_cert.py/send` (POST the CSR).

## Apache conf (SSL) (cont)

---

### SSL access control:

```
<Directory /var/www/html/https/rfiles>  
    SSLVerifyClient require  
    SSLRequire %{SSL_CLIENT_S_DN_CN} eq %{REMOTE_HOST}  
</Directory>
```

### along with:

```
SSLCertificateFile /.../server.crt  
SSLCertificateKeyFile /.../server.key  
SSLCACertificateFile /.../ca.crt
```

## Apache conf (rewrite)

---

Goal is to check, in this order, for a particular file in different directories.

Different component for this:  
a hostname to cluster map:

```
lxplus019.cern.ch lxplus  
lxplus020.cern.ch lxplus  
lxplus023.cern.ch lxplus
```

## Apache conf (rewrite) (cont)

### Simple denying filtering (example):

```
RewriteRule ^/rfiles.*$ - [F]
```

```
RewriteRule ^/files/.*/*.*/.* - [F]
```

**For** /files/foo→ /rfiles/cluster/host/foo:

```
RewriteCond %{DOCUMENT_ROOT}/rfiles/${NodesToClusters:\
```

```
%{REMOTE_HOST}|global}/%{REMOTE_HOST}/$1 -l [OR]
```

```
RewriteCond %{DOCUMENT_ROOT}/rfiles/${NodesToClusters:\
```

```
%{REMOTE_HOST}|global}/%{REMOTE_HOST}/$1 -f
```

```
RewriteRule ^/files/(.*)$ /rfiles/${NodesToClusters:\
```

```
%{REMOTE_HOST}|global}/%{REMOTE_HOST}/$1 [L]
```

### Same for :

- /files/foo→ /rfiles/cluster/foo:
- /files/foo→ /rfiles/foo:

## Server scripts

Already saw (part of) the certificate handling script. Another script is used for the preparation. Too long for a complete description:

- hostname as argument
- XML input (from machines' profiles) handled by SAX parser.
- simple script to search for specific file/script. Child scripts invoked with correct arguments.

Included with a custom login:

```
[hostname?]> lxplus001.cern.ch
Method type for passwd is script, scope: cluster
-> got ./scripts/lxplus/default/passwd/
Updating info for ../../rfiles/lxplus/passwd.tar.gz
-> OK
Method type for sshk is file, scope: node
Please put the file in [../../rfiles/lxplus/lxplus001.cern.ch/sshk.tar.gz]
Found 2 item(s)
```

## Client scripts

Two scripts:

- one for requesting a certificate
- one for requesting a file

Certificate script:

...

```
openssl genrsa -out key.out 1024
```

```
openssl req -new -batch -key key.out -out csr.pem
```

...

```
curl -s -f --cacert ca.crt \
```

```
    https://serv/CA/get_cert.py/send \
```

```
    -F "csr=@csr.pem;type=text/plain" -o node.crt
```

...

## Client scripts (cont)

---

### File request script:

```
curl -s --cacert ca.crt "https://serv/files/foo"\  
-E client.pem -o foo
```