SSH Tunneling examples with PoolTool.io

Version 0.4 - 18.04.2020

by @atada_stakepool (telegram)

First of all, we're using ssh in the first two scenarios. Please make sure that you know how to setup SSH with SSH-Keys. This is not going to work with normal password authentication, thx!

You need two machines for Scenario 1&2, most likely two linux machines.The one that is running your Stakepool-Node (with the IP Address you want to hide), and a second one like the PC of the Stakepool-Operator at home or a small RaspberryPI or a VM running in VirtualBox,etc.

I think it should work similar with Windows too, i haven't tested this yet. Mac should be working like Linux.

I have made a little sketch of the most common three scenarios how you could use ssh tunnels and also the Tor network to hide your real Stakepool-Node IP from being reported.

All Scenarios are having End-to-End encryption. The connection to api.pooltool.io is made via https, and additionally the ssh tunnel is encrypted too! :-)

Scenario 1

Node is running on a VPS/Cloudserver -> SSH-Server (Public_Node_IP, static for jormungandr)

*Operator is running a machine at home -> SSH-Client (Client_IP, **this can be a static or a dynamic one!**)*

Scenario 2 (other way around than Scenario 1)

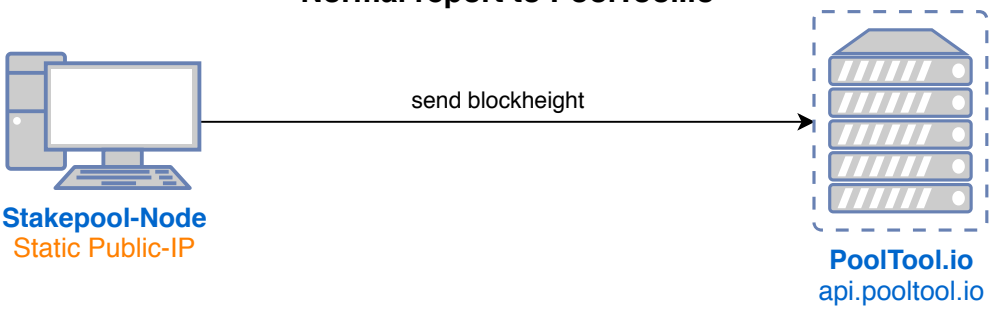
Node is running on a machine at home -> SSH-Client (Public_Node_IP, static for jormungandr)

*2nd machine is running somewere else -> SSH-Server (Server_IP,**static or dynamic-with-FQDN**)*

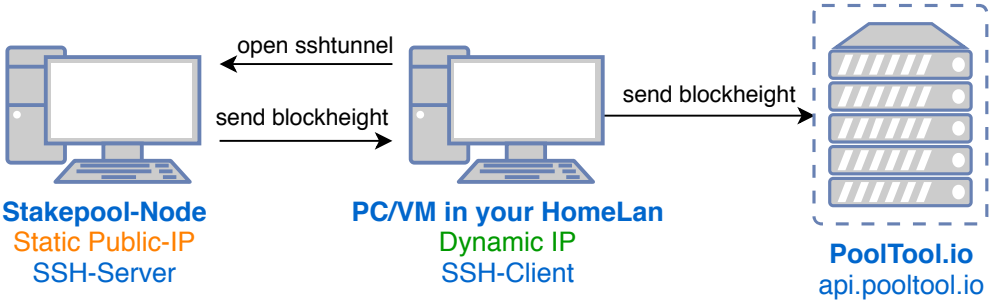
Scenario 3

Using the Tor network to hide your reporting IP via SOCKS Proxy.

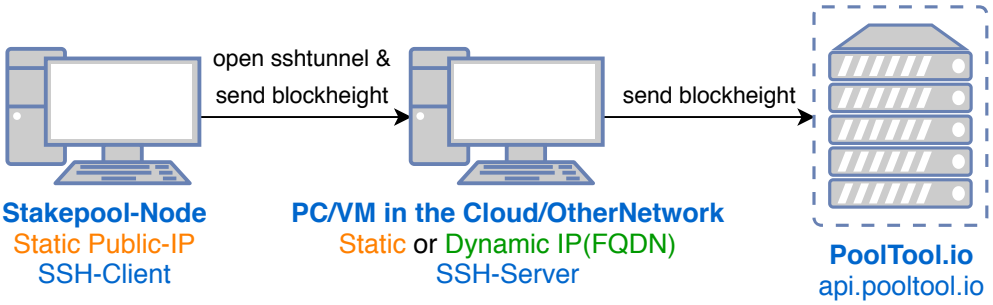
Normal report to PoolTool.io



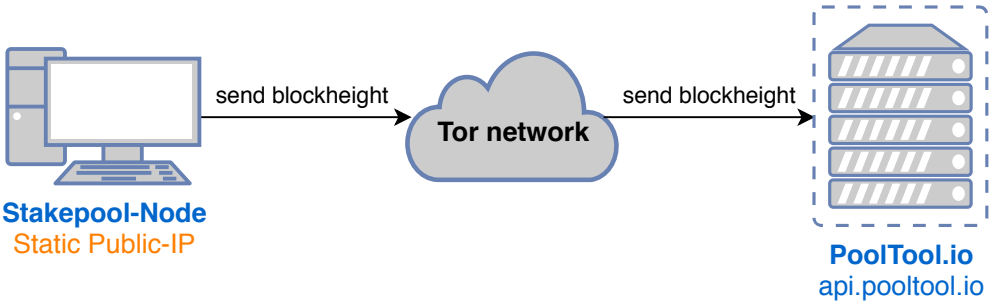
Scenario 1 - StakePool in the Cloud/VPS



Scenario 2 - StakePool at home



Scenario 3 - Using the Tor network



Scenario 1

Node is running on a VPS/Cloudserver -> SSH-Server (Public_Node_IP, static for jormungandr)

*Operator is running a machine at home -> SSH-Client (Client_IP, **this can be a static or a dynamic one!**)*

This is a common scenario for all that are running there node in the cloud/vps/vm etc. and having another Linux Machine (can be a Raspberry, LinuxVM etc) running at home/office.

Make sure that you can log into your SSH-Server (Stakepool machine) via ssh using ssh-keys from your SSH-CLIENT like:

```
CLIENT# ssh user@STAKEPOOLSERVERIP
```

The tip is sent to PoolTool via a https connection using curl in a shell script for example. We want that this is still a https connection in the future. So we have to make sure the requested FQDN is the same, in this case "api.pooltool.io". We now have to make the SSH-Tunnel from the Client to the Server. But we have to make a decision before that:

Do we want the Tunnelport on the Server to be in the range 1-1024, or is it ok that it is 1025-65535?

If we want it to be in the lower range, then the SSH connection from the SSH-CLIENT to the SSH-SERVER must be made via the root user on the SERVER like:

```
CLIENT# ssh root@STAKEPOOLSERVERIP
```

Otherwise if it is ok to use a higher portnumber, lets say **4443**, than you can use another user without root privilege like:

```
CLIENT# ssh cardano@STAKEPOOLSERVERIP
```

Lets go with the example of using Tunnelport **4443** and the user **cardano**

Please make sure that you can login with via SSH connection using SSH-Keys without a password!

We're using an "extended" version of the normal ssh program on the SSH-CLIENT, called autossh.

If autossh is not installed, you can install it like any other linux package f.e.:

```
CLIENT# sudo apt-get install autossh
```

On the SSH-SERVER side (Stakepool machine) you don't need any special program, a ssh server is still running. But you have to **add the following lines** to

SERVER: /etc/ssh/sshd_config

```
ClientAliveInterval 60

ClientAliveCountMax 2

AllowTcpForwarding yes

KeepAlive yes
```

After that restart your SSH Service like:

```
SERVER# sudo systemctl restart sshd
```

Now the fun part...

To open up the SSH-Tunnel in the background that is automatically reopened again if something occurs you just have to run the following command on the SSH-CLIENT (Operator Home Machine for example):

```
CLIENT# autossh -M 0 -f -o ConnectTimeout=10 -o
ServerAliveInterval=60 -o ServerAliveCountMax=2 -o
ExitOnForwardFailure=yes -p 22 -N -R 4443:api.pooltool.io:443
cardano@STAKEPOOLSERVERIP
```

Be sure to set the correct values for the following parameters: The first is the SSH Port number (you should not use the Standardport 22 normally) thats the value with the parameter -p 22.

The next parameter is the settings for the REVERSED ssh tunnel (-R) 4443:api.pooltool.io:443.

4443 is the port that we have choosen to be our Tunnelport, api.pooltool.io is the location were we want to redirect the Tunnel to. 443 is the destination port, in this case we want a https connection, so this has to be 443. And of course the user on the Stakepool-Server and the IP of the StakePool-Server.

Your tunnel is up! :-)

You should run this command for example via crontab and an @reboot entry so it automatically starts when your client-machine starts.

If you go to the SERVER and run

```
SERVER# netstat -tlpen
```

you should find an entry like:

```
tcp        0      0 127.0.0.1:4443          0.0.0.0:*               LISTEN      0          3994791826 27322/sshd: cardano
```

Ok, so now everything that you send to your localhost (127.0.0.1) on port 4443 will travel through the tunnel to your Client-Machine, and from there to api.pooltool.io:443.

What do we have to change on the server that this will work? We have to redirect the FQDN api.pooltool.io to 127.0.0.1 and change the port. To do so we **add the following line** to the hosts file:

SERVER: /etc/hosts

```
127.0.0.1       api.pooltool.io
```

The last step is to change the "curl call" in the sendmy_tip.sh script or any other script you use to send your Blockheight to Pooltool.

Edit your script and **add the Tunnel-Port-Number to the code**

curl ... "https://api.pooltool.io/v0/sharemytip?poolid=...

should be now

curl ... "https://api.pooltool.io:4443/v0/sharemytip?poolid=...

Thats it, you're now reporting to Pooltool via https connection through an encrypted ssh tunnel via your SSH-CLIENT (Home PC?) to PoolTool.

The IP address that PoolTool receives, is the IP address of the SSH-CLIENT and not your real STAKEPOOLSERVERIP! :-)

_____

Some thoughts: If you do not want to change any of the sending scripts, you have to use Tunnelport 443 and for that you have to make the SSH Tunnel via the root user as i wrote above. In that case you only have to change the /etc/hosts file on the Server. But it wouldn't ne possible to run for example another webserver with https on the Stakepool Server that also LISTEN on port 443.

_____

Next, Scenario 2 ...

Scenario 2 (other way around than Scenario 1)

Node is running on a machine at home -> SSH-Client (Public_Node_IP, static for jormungandr)

2nd machine is running somewere else -> SSH-SERVER (Server_IP,static or dynamic with FQDN)

This is very similar to the Scenario 1, the only difference is that you run your node at home and wanna hide that ip by using another machine out there with a different ip.

Make sure that you can log into your SSH-Server (another machine) via ssh using ssh-keys from your SSH-CLIENT (local node machine) like:

```
CLIENT# ssh remoteuser@SERVERIP
```

The tip is sent to PoolTool via a https connection using curl in a shell script for example. We want that this is still a https connection in the future. So we have to make sure the requested FQDN is the same, in this case "api.pooltool.io". At the moment this FQDN points to a few Pooltool Web IPs. We now have to make the SSH-Tunnel from the Client to the Server, before we have to make a decision:

Do we want the Tunnelport on the local machine to be in the Range 1-1024, or is it ok that it is in the range 1025-65535?

If we want it to be in the lower range, then the SSH connection from the SSH-CLIENT to the SSH-SERVER has to be made with root privilege like:

```
CLIENT# sudo ssh remoteuser@SERVERIP
```

Otherwise if it is ok to use a higher portnumber, lets say 4443, than you can use another user without root rights like:

```
CLIENT# ssh remoteuser@SERVERIP
```

Lets go with the example of using Tunnelport 443 (https) and the user remoteuser on the other machine that we want to connect to.

We're using an "extended" version of the normal ssh program on the SSH-CLIENT, called autossh.

If autossh is not installed, you can install it like any other linux package f.e.:

```
CLIENT# sudo apt-get install autossh
```

On the SSH-SERVER side (another machine) you don't need any special program, a ssh server is still running. But you have to **add the following lines** to

SERVER: /etc/ssh/sshd_config

```
ClientAliveInterval 60

ClientAliveCountMax 2

AllowTcpForwarding yes

KeepAlive yes
```

After that restart your SSH Service like:

```
SERVER# sudo systemctl restart sshd
```

To open up the SSH-Tunnel in the background that is automatically reopened again if something occurs you just have to run the following command on the SSH-CLIENT (node machine at home):

```
CLIENT# sudo autossh -M 0 -f -o ConnectTimeout=10 -o
ServerAliveInterval=60 -o ServerAliveCountMax=2 -o
ExitOnForwardFailure=yes -p 22 -N -L
127.0.0.1:443:api.pooltool.io:443 remoteuser@SERVERIP
```

Be sure to set the correct values for the following parameters: The first is the SSH Port number (you should not use the Standardport 22 normally) thats the value with the parameter -p 22.

The next parameter is the settings for the FORWARD ssh tunnel (-L) 127.0.0.1:443:api.pooltool.io:443. This is now a **forwarding** tunnel, because we will forward a connection on a local port via the remoteserver to another machine. Thats the other way around than Scenario 1!

127.0.0.1:443 is the **local** port that we have choosen to be our Tunnelport, api.pooltool.io:443 is the location were we want to redirect the Tunnel to. In this case we want a https connection, so this has to be :443. And of course the user 'remoteuser' on the Remote SSH-Server-IP we wanna use.

You see that in this example i used the command with **sudo** infront, because we wanna open a local portnumber below 1024, so the command is running with root rights.

Your tunnel is up! :-)

You should run this command for example via crontab and an @reboot entry so it automatically starts when your client-machine starts. Run with specified user root in this example.

If you now run the following command on the local node machine

```
CLIENT# netstat -tlpen
```

you should find an entry like:

```
tcp      0     0 127.0.0.1:443       0.0.0.0:*          LISTEN    0       25874     2115/ssh
```

Ok, so now everything that you send to your localhost (127.0.0.1) on port 443 will travel through the tunnel to your Server-Machine (SERVERIP), and from there to api.pooltool.io:443.

What else do we have to change on the client (node machine) that this will work? We have to redirect the FQDN api.pooltool.io to 127.0.0.1. To do so we **add the following line** to the hosts file:

CLIENT: /etc/hosts

```
127.0.0.1       api.pooltool.io
```

Thats it, you're now reporting to Pooltool via https connection through an encrypted ssh tunnel via your SSH-SERVER (another machine out on the internet) to PoolTool.

The IP address that PoolTool receives, is the IP address of the SSH-SERVER and not your real local STAKEPOOL IP! If you use a FQDN for your SSH-SERVER out there, than the ip can also be dynamic like:

```
CLIENT# sudo autossh -M 0 -f -o ConnectTimeout=10 -o
ServerAliveInterval=60 -o ServerAliveCountMax=2 -o
ExitOnForwardFailure=yes -p 22 -N -L
127.0.0.1:443:api.pooltool.io:443 remoteuser@littleserver.dyndns.org
```

Scenario 3

Using the Tor network to hide your reporting IP via SOCKS Proxy.

First, we have to install Tor on the system (Stakepool machine). If you're using Debian, you just can install Tor with:

```
SERVER# sudo apt install tor
```

If you're using Ubuntu or you want the latest version for your system, you should follow the installation guide on the Tor-Project website:

```
https://2019.www.torproject.org/docs/debian.html.en
```

But to keep it short, you add the Tor-Project repository to your sources list and after that you do a normal apt/yim install of Tor. You can also find instructions for MacOS (using brew) at the link above.

Normally after you've installed Tor on your machine, the service should run automatically. Just make sure to execute the following commands to enable the service autostart at boot:

```
SERVER# sudo systemctl enable tor && sudo systemctl start tor
```

If you wanna run Tor as a Daemon instead, please edit the config file (/etc/tor/torrc) and enable the line RunAsDaemon like:

```
SERVER: /etc/tor/torrc

RunAsDaemon 1
```

After that, you can start Tor as a daemon process with a simple

```
SERVER# tor
```

If you don't want to set this in the config file, you can also start Tor with the following commandline:

```
SERVER# tor --runasdaemon 1
```

Ok, your Tor is now running. As a system service or as a daemon. Make sure to start it at boottime if you run it as daemon via @reboot in crontab! When you run the following command

```
SERVER# netstat -tlpen
```

you should see an entry like:

```
tcp      0     0 127.0.0.1:9050       0.0.0.0:*          LISTEN     0       18472     504/tor
```

This shows the listening SOCKS port of Tor, we're almost done. The last step is to change the "curl call" in the sendmy_tip.sh script or any other script you use to send your Blockheight to Pooltool.

Edit your script and **add the Socks5-Hostname parameter to the code**

```
curl ... "https://api.pooltool.io/v0/sharemytip?poolid=...
```

should be now

```
curl ... --socks5-hostname localhost:9050 "https://api.pooltool.io/v0/sharemytip?poolid=...
```

Done! You're now reporting to Pooltool via the Tor network, your IP should be different every few minutes. :-)