

TOAD - SSH Tunnelling

Norman [Norm] Dunbar

Table of Contents

1. Introduction	1
2. The General Process.....	1
3. Edit Tnsnames.ora	2
4. Convert SSH Key Pairs	3
4.1. MobaXterm to Putty	3
4.2. MobaXterm or Putty to Bitvise	4
4.3. Bitvise to Putty.....	4
4.4. Bitvise to MobaXterm	5
5. Tunnelling - The Simple Way	5
5.1. Tunnelling with Putty	5
5.2. Tunnelling with MobaXterm	7
5.3. Tunnelling with BitVise.....	8
6. Tunnel Testing	9

1. Introduction

Sometimes, it happens, that you cannot access a database from your desktop and this makes using Toad quite difficult. I have worked at a number of locations where there are in-house databases some of which are accessible directly from the desktop, some are only accessible from a "jumping off" server which we need to SSH onto, and from there, run SQL*Plus to access the database(s), and some databases "in the cloud". In this case, the cloud means Amazon's AWS or Oracle's OCI. None of what follows has been tested on Microsoft's Azure cloud, although I have set up some databases there as well.

The decisions to prevent direct access to the databases, from the desktop, were made based on a security decision - *it is far more secure if the users cannot access the database*. As you know, these sort of decisions are usually made to make life difficult for use DBAs! <Wink>

As ever, where there's a will, there's a way!

The remainder of this document describes the process required to access, via an SSH tunnel, databases we might not normally be able to access from our desktop. There are three separate products used:

- Putty - available for free from <https://www.putty.org/>. (There's also a link to BitVise SSH Client here as well!)
- MobaXterm - able for home use at <https://mobaxterm.mobatek.net/download.html>. There are restrictions though - limits on the number of profiles you can save, plus only three saved SSH Tunnels.
- BitVise SSH Client - freely available from <https://www.bitvise.com/ssh-client-download>. You don't need the server version, only the client.

To paraphrase Shakespeare, *read on MacDuff*.

2. The General Process

As we already have an SSH server, let's call it **CLOUD_SSH** (because that's its name!) and assume it has been correctly set up in DNS with an IP address and all the rest, which we *must* use to access the cloud databases, we are already half way there. It is also assumed that we are using a private/public key pair to access the SSH server in question - no passwords are required. In my own particular case, I used MobaXterm, so my SSH keys were created using 'tools → MobaKeyGen' then creating and saving a private and public key pair.

The process is almost identical to that used by Putty, where the separate application is named **puTTYgen**.

If a database can be accessed from the **CLOUD_SSH** server, then it is a potential candidate for access from your desktop. Let us assume we have the following databases which we would like to connect to from within Toad:

- A development database, **AWSDEV**, in Amazon AWS, running on a host named **dev.aws.cloud.com**;

- A system test database also in AWS, **AWSSIT**, running on a host named **sit.aws.cloud.com**;
- A production database in Oracle Cloud, **OCIPROD**, running on a primary host named **prod.oci.cloud.com** with a standby instance on host **stby.oci.cloud.com**;
- All database listeners are running on the standard port, **1521**.

As there are four databases on different servers, we will need to consider four local ports to be used for the tunnelling. I've decided on:

- Port **1234** for the **AWSDEV** database;
- Port **1235** for the **AWSSIT** database;
- Port **1237** for the **OCIPROD** primary database;
- Port **1238** for the **OCIPROD** standby database.

3. Edit Tnsnames.ora

Now that we know the servers and ports we need to access, and the local ports we will use to tunnel through to them, we need to edit our **tnsnames.ora** file (or equivalent) so that we can route Toad etc to the local ports rather than the inaccessible remote server ports.

Edit **tnsnames.ora** to add a new entry for each of the databases. The host and port will be **localhost** and **1234** as follows:

```

awsdev =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1234))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = AWSDEV)
    )
  )

awssit =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1235))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = AWSSIT)
    )
  )

ociprod =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1237))
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1238))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = OCIPROD)
    )
  )

```

4. Convert SSH Key Pairs

If your key pair was created with one particular utility, and you need to use another, then this section should help.

My keys were created in MobaXterm, so I have to convert them for use with the other two utilities.

4.1. MobaXterm to Putty

- Open the **puttygen** utility;
- Click the 'Load' button to 'Load an existing private key';
- Change the file type filter to 'all files' and navigate to where your MobaXterm keys are saved;
- Select the appropriate key - **id_rsa** in my case;
- Accept the prompt that appears when the import succeeds;
- Click 'Save Private Key';

- Click 'Yes' to save without a pass-phrase;
- Navigate to a suitable location, enter a file name with a 'ppk' extension, and click 'Save';
- Close **puttygen**;

4.2. MobaXterm or Putty to Bitvise

- Open the Bitvise SSH client tool, and on the 'Login' tab, around the middle, you should see a link entitled 'Client key manager'. Click it.

On the dialog that opens:

- Click on the 'Import' button;
- Change the file type filter to 'all files' (for MobaXterm keys) or 'Putty private key files' (for Putty) then navigate to where MobaXterm/Putty has saved your keys - you chose it at creation time, so you should know where it is!
- Select the *private* key file, **id_rsa** for example, and click 'Open'.

On the dialogue that opens:

- Change the 'Location' dropdown to select 'Global' (at the top), rather than 'Profile';
- Make a note of the global number that is generated, the first key imported will be 'Global 1'. You need this later when running the command line utility to open the tunnels.
- Change the comment to read something meaningful, such as "Imported MobaXterm/Putty SSH Private Key" and click 'Import'.
- Close the dialogue.

4.3. Bitvise to Putty

- Open the Bitvise SSH client tool, and on the 'Login' tab, around the middle, you should see a link entitled 'Client key manager'. Click it.

On the dialogue that opens:

- Click the desired key in the grid at the top;
- Click on the 'Export' button;
- Choose to export the *private* key and select for Putty;
- Click 'Export using empty passphrase' then click on 'Continue' on the pass-phrase dialogue, if it appears, we are not using pass-phrases;
- Enter a filename and click 'Save'.
- Close the key manager.

4.4. Bitwise to MobaXterm

- Open the Bitwise SSH client tool, and on the 'Login' tab, around the middle, you should see a link entitled 'Client key manager'. Click it.

On the dialogue that opens:

- Click the desired key in the grid at the top;
- Click on the 'Export' button;
- Choose to export the *private* key and select for OpenSSH;
- Click 'Export using empty passphrase' then click on 'Continue' on the pass-phrase dialogue, if it appears, we are not using pass-phrases;
- Enter a filename and click 'Save'. There can be any extension that you wish. The file type filter here is always 'all files';
- Close the key manager.

5. Tunnelling - The Simple Way

As with much in life, there's an easy way and a harder way to do things. This is the easy way.

Once you have downloaded your desired SSH client software, see [Introduction](#) above, sorted out your key pairs ([Convert SSH Key Pairs](#) above) and placed the executable's folder on your `%PATH%`, all that is required to do is as follows:

5.1. Tunnelling with Putty

5.1.1. The Quick but Tedious Way!

```
putty -N -L 1234:dev.aws.cloud.com:1521 oracle_user@CLOUD_SSH
putty -N -L 1235:sit.aws.cloud.com:1521 oracle_user@CLOUD_SSH
putty -N -L 1237:prod.oci.cloud.com:1521 oracle_user@CLOUD_SSH
putty -N -L 1238:stby.oci.cloud.com:1521 oracle_user@CLOUD_SSH
```

Each one will open up a terminal window, but you cannot type into it, it is merely acting as a tunnel for the local ports.

There are now 4 separate tunnels running to the required database servers. Each tunnel goes from the local port - `123n`, logs on as the `oracle_user` on the `CLOUD_SSH` server (the jumping off box), and from there to the appropriate database server - `dev.aws`, `sit.aws`, `prod.oci` or `stby.oci` - on port `1521`.

If you now attempt to `tnsping awsdev` or any of the others, you should see a result back from the listener. If so, you can now open Toad and configure a new connection to the remote database but using the `tnsnames.ora` entries that connect to `localhost` instead.

5.1.2. The Slow but Better way!

Running a separate session for each tunnel you want to create gets a bit tedious after a while and also, fills up the screen with unusable ssh session windows. Why not create a single putty session, name it SSH_TUNNELS, and combine all your tunnels into one session?

- Start putty as normal;
- On the Connection → ssh → Tunnels dialogue, add your required tunnels, one by one, by:
 - Filling in the local port (1234, 1235, 1236 etc) in the Source Port edit field;
 - Filling in the destination host and port (dev.aws.cloud.com:1521 etc) in the Destination edit field;
 - Clicking the **Add** button;
- On the SSH → Auth dialogue page, click the **Browse** button and navigate to, and select your *private* putty keyfile;
- On the Session tab, fill in the user_name and Host Name for the jump-off box you are using (oracle_user@CLOUD_SSH);
- Make sure the session type is **SSH**.
- Type the name, SSH_TUNNELS, in the Saved Sessions edit field;
- Click the **Save** button.

Now you simply click the **open** button and one screen will appear and announce that it is 'authenticating with public key "whatever"' - but not much else. You can check if the tunnel(s) are open by opening a command window and:

```
netstat -a | find "123"
```

Proto	Local Address	Foreign Address	State
...			
TCP	127.0.0.1:1234	MY_PC_NAME:0	LISTENING
TCP	127.0.0.1:1235	MY_PC_NAME:0	LISTENING
TCP	127.0.0.1:1237	MY_PC_NAME:0	LISTENING
TCP	127.0.0.1:1238	MY_PC_NAME:0	LISTENING
...			

On subsequent occasions, all you have to do is run this in a batch file, desktop shortcut or a command line session:

```
putty -load SSH_TUNNELS
```

and all the tunnels you added will be up and running for you.

5.2. Tunnelling with MobaXterm

Open MobaXterm, and click on the 'home' tab - the one with the little house on it. You will have a choice, possibly, to 'start local terminal' or, if appropriate, to 'recover previous sessions' - if your Windows box crashed while MobaXterm was connected and running sessions. Start a local terminal.

When the terminal starts, type the following:

```
ssh -N -L 1234:dev.aws.cloud.com:1521 oracle_user@CLOUD_SSH &  
ssh -N -L 1235:sit.aws.cloud.com:1521 oracle_user@CLOUD_SSH &  
ssh -N -L 1237:prod.oci.cloud.com:1521 oracle_user@CLOUD_SSH &  
ssh -N -L 1238:stby.oci.cloud.com:1521 oracle_user@CLOUD_SSH &
```



You could, if you intend to run this again, save those commands in a text file named, for example, `tunnels.sh` and make it executable with `chmod ug+x tunnels.sh`, next time you need to run it, start a local session and:

```
./tunnels.sh
```

That starts 4 separate tunnels to the required database servers. Each tunnel goes from the local port - `123n`, logs on as the `oracle_user` on the `CLOUD_SSH` server (our jumping off box), and from there to the appropriate database server - `dev.aws`, `sit.aws`, `prod.oci` or `stby.oci` - on port `1521`.

These jobs run in the background, so to determine the job numbers for later, you can type the command:

```
jobs
```

The result will appear similar to:

```
[1]+  Running                  /bin/ssh.exe -N -L 1234:dev.aws.cloud.com:1521  
oracle@10.128.3.242 &  
...
```

The number in square brackets is the background job number, which you can kill off as follows:

```
fg 1  
CTRL-C
```

Repeat for all the background jobs running tunnels. (Or just close the local session!)

If you now attempt to `tnsping awssit` or any of the others, you should see a result back from the listener. If so, you can now open Toad and configure a new connection to the remote database but

using the `tnsnames.ora` entries that connect to `localhost` instead.

5.3. Tunnelling with BitVise

This SSH Client is quite different from the others, it does have a command line version which opens tunnels and you can do this interactively, or using a parameter file. The easy method is using a parameter file.

Open a text editor and type the following, replacing my ports, database servers and ssh logins as befits your installation.

```
127.0.0.1,1234,dev.aws.cloud.com,1521
127.0.0.1,1235,sit.aws.cloud.com,1521
127.0.0.1,1237,prod.oci.cloud.com,1521
127.0.0.1,1238,stby.oci.cloud.com,1521
```



The format is local-interface, local-port, desination-host, destination-port and you cannot use `localhost` for the local-interface, it must be a valid IP Address - 127.0.0.1 is IPv4 for Localhost.

Save the file, for example, `c:\users\your_name\tunnels.txt`.

The command to run the 4 tunnels is:

```
stnlc oracle_user@CLOUD_SSH -pk=1 -c2sFile=c:\\users\\your_name\\tunnels.txt
```



For some reason, even though this is a Windows application, the directory separators in the path to the `tunnels.txt` file *must* be doubled up. If not, you get an error message saying that the file cannot be found.

The `-pk=1` indicates the 'Global 1' key that you generated or imported above. Make sure you use the correct id number. Also, if you inadvertently saved the key as 'Profile' rather than 'Global', then use `-pk=p1` if your key is 'Profile 1'.

This is a *slightly* interactive utility. Any time there is traffic on any of the opened tunnels, details will be written to the console that opened when you executed the above command. You exit, closing the tunnels, using the `quit` command.

Alternatively, if you add `-unat=y` to the command line above, you get unattended operation - in which case, CTRL-C aborts and closes the tunnels, and there is no logging on screen, of the tunnels' usage.

If you now attempt to `tnsping ociprod` or any of the others, you should see a result back from the production database listener. If so, you can now open Toad and configure a new connection to the remote database but using the `tnsnames.ora` entries that connect to `localhost` instead.

6. Tunnel Testing

Once the tunnel is open, and your `tnsnames.ora` is up to date with the local ports you are using, test:

```
tnsping awsdev
```

```
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST =  
localhost)(PORT = 1234)) (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME =  
AWSDEV)))  
OK (180 msec)
```

I call that a result. You can now use Toad, SQL*Plus etc to connect directly to the databases, from your desktop, without having to mess about logging onto the 'jump-off' box - which probably doesn't have the tools you use daily for everything else.

Enjoy.

Norm. [TeamT]