

# Solution for SSH Unable to Negotiate Errors

---

 [infosecmatter.com/solution-for-ssh-unable-to-negotiate-errors](https://infosecmatter.com/solution-for-ssh-unable-to-negotiate-errors)

July 27, 2020

This is a quick howto post that can help not only penetration testers, but also network engineers, administrators and other specialists who work with network equipment, old storages, embedded systems, kiosk devices, and variety of other legacy systems with SSH interface (tcp/22).

We are going to talk about SSH compatibility issues and those pesky SSH unable to negotiate login errors, No matching host key found errors and more. And in the end we will present solution for fixing all of these errors once and for all.

## Introduction

---

Since this website is primarily about pentesting, I'm going to describe the issue from a perspective of a penetration tester.

One of the typical activities performed during infrastructure penetration tests are login attacks against SSH interfaces (aka. SSH login bruteforcing).

The idea behind SSH login bruteforcing is to identify systems and devices in the network configured with weak and/or default credentials.

There always happen to be some systems in the network that can be broken into this way. And this is true even for mature environments. Pentesters typically find these cases using automated tools such as:

- Metasploit
- Medusa
- Hydra
- Nmap
- ...

Now, the larger the environment, the higher is the number of valid credentials that are usually found. The following list is definitely not a rarity to find in a mid-size or larger environment:

```

kali@kali:~/ssh.bruteforce$ cat * | grep -i success
ACCOUNT FOUND: [ssh] Host: 10.4.0.120 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.0.222 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.18.210 User: admin Password: password [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.18.212 User: admin Password: password [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.22.10 User: root Password: root [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.22.16 User: root Password: root [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.22.19 User: root Password: root [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.65 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.66 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.66 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.67 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.67 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.68 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.68 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.69 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.69 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.73 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.73 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.76 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.77 User: root Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.50.78 User: admin Password: P@ssw0rd [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.51 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.51 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.54 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.54 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.55 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.55 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.60 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.4.8.60 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.113 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.113 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.115 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.115 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.118 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.118 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.119 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.119 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.166 User: admin Password: password [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.167 User: admin Password: password [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.64 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.64 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.65 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.65 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.67 User: admin Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.67 User: root Password: 12345678 [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.90 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.91 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.92 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.5.21.93 User: admin Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.9.202.101 User: root Password: admin [SUCCESS]
ACCOUNT FOUND: [ssh] Host: 10.9.202.103 User: diag Password: admin [SUCCESS]

```

But before reporting any of these findings, we should first verify them. Some of the results might be false positives and it would also look nice in the report if we managed to identify what all those systems and devices are.

So how do we do that? I don't know how you do it, but I typically manually inspect every single instance using the standard ssh client (OpenSSH) that is available in my system (usually Kali Linux) and examine each case by hand.



## The problem

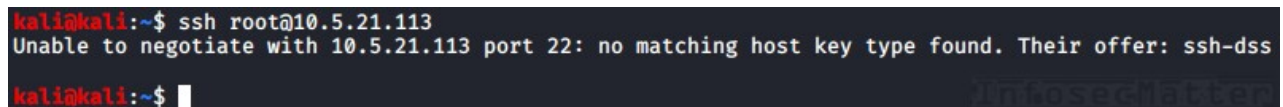
---

But the problem is this: Many times the SSH client fails to connect to the remote system due to a SSH negotiation error. Some of the typical error messages are:

- No matching cipher found. Their offer: aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc
- No matching key exchange method found. Their offer: diffie-hellman-group1-sha1
- No matching host key type found. Their offer: ssh-dss
- DH GEX group out of range

For example:

```
# ssh root@10.5.21.113
Unable to negotiate with 10.5.21.113 port 22: no matching host key type found.
Their offer: ssh-dss
```

A terminal window showing the command 'ssh root@10.5.21.113' being executed. The output is 'Unable to negotiate with 10.5.21.113 port 22: no matching host key type found. Their offer: ssh-dss'. The prompt is 'kali@kali:~\$'. There is a watermark 'infosecmatter' in the bottom right corner.

```
kali@kali:~$ ssh root@10.5.21.113
Unable to negotiate with 10.5.21.113 port 22: no matching host key type found. Their offer: ssh-dss
kali@kali:~$
```

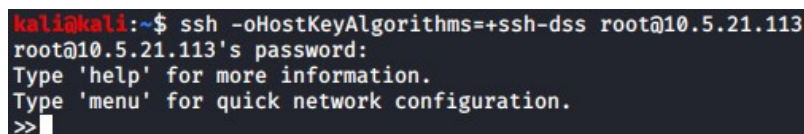
Why these errors happen? In most cases this is because we are connecting to legacy devices with outdated cryptography support – old Cisco routers, network equipment, outdated appliances and so on.

These legacy devices only support weak and outdated cipher suites, legacy key exchange methods etc. And our ssh client doesn't trust those old methods any more (by default), so we cannot connect.

So what do we do?

Good news is that we can typically always find a workaround (solution) on the Internet which solves our particular problem connecting to the device – typically by adding an extra option to the ssh client on the command line:

```
# ssh -oHostKeyAlgorithms=+ssh-dss root@10.5.21.113
```

A terminal window showing the command 'ssh -oHostKeyAlgorithms=+ssh-dss root@10.5.21.113' being executed. The output is 'root@10.5.21.113's password:', 'Type \'help\' for more information.', and 'Type \'menu\' for quick network configuration.' The prompt is 'kali@kali:~\$'. There is a watermark 'infosecmatter' in the bottom right corner.

```
kali@kali:~$ ssh -oHostKeyAlgorithms=+ssh-dss root@10.5.21.113
root@10.5.21.113's password:
Type 'help' for more information.
Type 'menu' for quick network configuration.
>>
```

But this approach is tiresome and soon can become exhausting, especially if we have many systems to check and if there are many different devices throwing different errors.

Let's review some of these typical problems and their workarounds, before jumping to the solution.

## SSH login error examples

---

Here are some of the typical SSH login compatibility issues with their workarounds.

## No matching key exchange method found. Their offer: diffie-hellman-group1-sha1

---

Typical SSH error message:

```
# ssh admin@10.200.180.62
Unable to negotiate with 10.200.180.62 port 22: no matching key exchange method
found. Their offer: diffie-hellman-group1-sha1
```

Workaround (found [here](#)):

```
# ssh -oKexAlgorithms+=diffie-hellman-group1-sha1 admin@10.200.180.62
```

## No matching cipher found. Their offer: aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc

---

Typical SSH error message:

```
# ssh root@192.168.2.44
Unable to negotiate with 192.168.2.44 port 22: no matching cipher found. Their
offer: aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc
```

Workaround (found [here](#)):

```
# ssh -c aes256-cbc root@192.168.2.44
```

## No matching host key type found. Their offer: ssh-dss

---

Typical SSH error message:

```
# ssh root@192.168.2.100
Unable to negotiate with 192.168.2.100 port 22: no matching host key type found.
Their offer: ssh-dss
```

Workaround (found [here](#) or [here](#)):

```
# ssh -oHostKeyAlgorithms+=ssh-dss root@192.168.2.100
```

## DH GEX group out of range

---

Typical SSH error message:

```
# ssh admin@10.2.100.41
ssh_dispatch_run_fatal: Connection to 10.2.100.41 port 22: DH GEX group out of
range
```

Workaround (found [here](#) or [here](#)):

```
# ssh -o KexAlgorithms=diffie-hellman-group1-sha1,curve25519-
sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-
hellman-group-exchange-sha256,diffie-hellman-group14-sha1 admin@10.2.100.41
```

Sometimes these workarounds solve one problem only to discover that there is yet another / different problem afterwards which we have to google again and add another command line option.

But no more! Here's how we can solve all these SSH login negotiation problems once and for all.

## Ultimate SSH login solution (fix everything)

---

To fix all these SSH compatibility issues, we can simply temporarily enable all legacy cryptography methods that our SSH client currently supports. That means enable all:

- Message authentication codes (MACs)
- Key exchange methods
- Host key algorithms
- Ciphers

To get the list of all supported algorithms, ciphers and methods that our SSH client currently supports, we can use the '-Q' option like this:

```
ssh -Q mac
ssh -Q kex
ssh -Q key
ssh -Q cipher
```

For example:

A terminal window with a dark background. The prompt is 'kali@kali:~\$'. The command 'ssh -Q mac' has been entered. The output lists 14 MAC algorithms: hmac-sha1, hmac-sha1-96, hmac-sha2-256, hmac-sha2-512, hmac-md5, hmac-md5-96, umac-64@openssh.com, umac-128@openssh.com, hmac-sha1-etm@openssh.com, hmac-sha1-96-etm@openssh.com, hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com, hmac-md5-etm@openssh.com, hmac-md5-96-etm@openssh.com, umac-64-etm@openssh.com, and umac-128-etm@openssh.com. The prompt 'kali@kali:~\$' is visible again at the bottom.

```
kali@kali:~$ ssh -Q mac
hmac-sha1
hmac-sha1-96
hmac-sha2-256
hmac-sha2-512
hmac-md5
hmac-md5-96
umac-64@openssh.com
umac-128@openssh.com
hmac-sha1-etm@openssh.com
hmac-sha1-96-etm@openssh.com
hmac-sha2-256-etm@openssh.com
hmac-sha2-512-etm@openssh.com
hmac-md5-etm@openssh.com
hmac-md5-96-etm@openssh.com
umac-64-etm@openssh.com
umac-128-etm@openssh.com
kali@kali:~$
```

And now all we have to do is to re-format it a bit and put it into our SSH client configuration file in our HOME folder `~/.ssh/config`.

So, here is **the ultimate fix for all SSH login negotiation errors**:

```
{
echo -n 'Ciphers '
ssh -Q cipher | tr '\n' ',' | sed -e 's/,,$//'; echo

echo -n 'MACs '
ssh -Q mac | tr '\n' ',' | sed -e 's/,,$//'; echo

echo -n 'HostKeyAlgorithms '
ssh -Q key | tr '\n' ',' | sed -e 's/,,$//'; echo

echo -n 'KexAlgorithms '
ssh -Q kex | tr '\n' ',' | sed -e 's/,,$//'; echo

} >> ~/.ssh/config
```

The command above will add the following options to your SSH client configuration:

```
Ciphers 3des-cbc,aes128-cbc,aes192-cbc,aes256-cbc,rijndael-
cbc@lysator.liu.se,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-
gcm@openssh.com,chacha20-poly1305@openssh.com
MACs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,umac-
64@openssh.com,umac-128@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha1-96-
etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-
md5-etm@openssh.com,hmac-md5-96-etm@openssh.com,umac-64-etm@openssh.com,umac-128-
etm@openssh.com
HostKeyAlgorithms ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,ssh-rsa,ssh-
dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-rsa-cert-
v01@openssh.com,ssh-dss-cert-v01@openssh.com,ecdsa-sha2-nistp256-cert-
v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-
v01@openssh.com
KexAlgorithms diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-
hellman-group14-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-
sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group-exchange-
sha256,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,curve25519-
sha256,curve25519-sha256@libssh.org,sntrup4591761x25519-sha512@tinyssh.org
```

Depending on your particular SSH client version it may look a little different, but the point is that it will enable ALL methods and ciphers that your current SSH client supports.

Now we should be able to connect to any legacy device or system on the network without any problem.

## Final words

---

Make sure that this arrangement is only temporary – do not ever use this on a production system as a long term solution. Make sure to revert the changes once you don't need them any more.

The solution presented here works by enabling all ciphers, key exchange methods and algorithms that were disabled in our OpenSSH client due to their security flaws and weaknesses.

In case this solution still hasn't fixed your particular SSH negotiation login problem, you simply have to get another (older) version of the OpenSSH client or get another SSH client.

If this post was useful for you and you would like more tips like this, please [subscribe](#) to our mailing list and follow us on [Twitter](#) and [Facebook](#) and get notified about new additions!

## SHARE THIS

**TAGS** | [Brute force](#) | [Cisco](#) | [Firewall](#) | [Kali Linux](#) | [Legacy devices](#) | [Login attack](#) | [Negotiation](#) | [OpenSSH](#) | [Router](#) | [SSH](#) | [Switch](#) | [Workaround](#)

---