# **A top-down introduction to SSH and how it enables secure data-sharing**

### **What is SSH?**

SSH is short for ‘secure shell’. It is a protocol for sharing data between two computers over the internet. A protocol is essentially a set of rules that define the language that computers can use to communicate.

### **How is a SSH session established?**

There are several processes that need to happen between two computers in order for an SSH session to begin.

1. First we need a way of setting up a secure method of exchanging messages between the computers. We need to set up an **encrypted channel.**
2. We need a way of checking that the data received by the host hasn’t been tampered with. This is called **verification** and here we are verifying the integrity of the data that is sent by the client.
3. **Verification (again)**. We need a way of checking that the computer we are communicating with isn’t an imposter. This is another form of verification but here we are verifying the identity of the computer.

After these steps, we can share ‘secret’ data securely and we can also check if a client has permission to access a host. Each of these sections below will go into more detail on these steps.

### **Setting up an encrypted channel**

A core part of the SSH protocol is that is it secure (it is in even in the name!), meaning all information that is sent using SSH is encrypted.

#### **How does this information get encrypted?**

Encrypting essentially just means ‘jumbling up the letters’ using some clever maths. Both computers need to have away of encrypting the information so that only the other computer can decrypt the information and understand it.

#### **How does this work?**

Both computers have an identical version of a **symmetric key.** The symmetric key is just a string of letters stored somewhere on the computers. The computers can use the symmetric keys to encrypt and also decrypt messages sent to them.

Using this symmetric key approach is called **symmetric encryption.** The ‘symmetric’ part comes from the fact the symmetric key on each computer is identical. This approach works really well … but it only works as long as no other computers have access to the symmetric key.

#### **Generating Symmetric Keys**

Both computers each have their own private key and public key. Together they form a **key-pair**. The computers **share their public keys** with each other over the internet. So, at this point in the process each computer knows 1.its own private key, 2.its own public key, and 3.the other computer’s public key.

Both computers then use these 3 pieces of information to independently generate an **identical** symmetric key. Each computer uses a mathematical algorithm which uses the 3 inputs mentioned above. This algorithm is part of the Diffie-Hellman key exchange algorithm.

The important thing to take away here is that computers have **shared only public information** over the internet **but have still been able to create symmetric keys!** The approach of using key-pairs and sharing public information to generate identical symmetric keys is called **asymmetric encryption**. It is called ‘asymmetric’ because both computers start off with their own, different, key pairs.

### **Verification**

So we can communicate securely. But the next part of the process of establishing an SSH session is to verify that the data hasn’t been tampered with as it has been transmitted **and** that the other computer is actually who it says it is.

#### **Hashing**

We have to use a **hash** function. This is just a mathematical function that takes inputs and produces a string of a fixed size. The important feature of this function is that it is virtually impossible to work out what the inputs were just using the outputs.

After a client and a host have generated their symmetric keys, the client will use a hashing function to generate a HMAC. This just stands for “hash-based message authentication code”. This is just another string of characters/numbers. The client will send this HMAC to the server for verification.

The ingredients to the hashing function are

* The symmetric key on the client
* The package sequence number (each message that is sent is contained in a ‘package’ of information)
* The (encrypted!!!) message contents

#### **How does the host use this information?**

When the host receives the HMAC, it can use **the same** hash function with these three ingredients:

* its own copy of the (identical!) symmetric key,
* the package sequence number,
* and the encrypted message.

If the hashed value it computes is the same as the HMAC it received from the client, then we have verified that the connecting computer is the same as the computer who has the symmetric key.

Remember that only the host and client know what the symmetric key is and no other computers do!

So here it doesn’t matter that the host doesn’t know the decoded contents of the encrypted message —the host has still verified the identity of the connecting computer!

The beauty of this approach is that we have not just verified the identity of the client and made sure that the data hasn’t been tampered, but we have done so securely (**without sharing any private information)**.

### **Authentication**

The client can send the host an (encrypted) message containing a password. The host can decrypt the message and check the password in a database to check if the client has permission to access the specified ‘user’ (area of the computer). Job done.

### **Conclusion** SSH is an important tool used to remotely control other computers.

SSH is secure because both computers can encrypt and decrypt message using identical symmetric keys (known as ‘symmetric encryption’). We can then use SSH to securely send data between the computers.