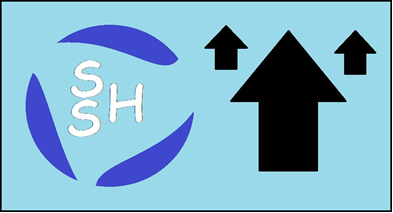
|  |
| --- |
| DCF255NCC – Professor Najma Ismat |
| THE WORKINGS OF THE SSH PROTOCOL |
| FALL 2019 – GROUP G2 |

|  |
| --- |
|  |

****

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| Stuart Evans | 106452188 |
| Jin Rong Huang | 114648181 |
| Navid Najarali Vadsariya | 136884186 |

**Table of Contents**

[**What is It?** 1](#_Toc26378874)

[**Why is It Necessary?** 1](#_Toc26378875)

[**The History of SSH: How Did it Come to Be?** 2](#_Toc26378876)

[**How Does It Work?** 2](#_Toc26378877)

[Symmetric (or shared key) Encryption 3](#_Toc26378878)

[Asymmetric Encryption 3](#_Toc26378879)

[One-Way Hashing 4](#_Toc26378880)

[**Specifications** 5](#_Toc26378881)

[**Additional Information** 5](#_Toc26378882)

[**To Reiterate: Why Is SSH Useful?** 5](#_Toc26378883)

[**In Conclusion** 6](#_Toc26378884)

[**Bibliography** i](#_Toc26378885)

# **What is It?**

SSH, which stands for Secure Shell, is a protocol that governs connecting machines over potentially unsecured networks such as the Internet. Programmers often need to access systems from a physical distance. An administrator, for instance, benefits from being able to connect their computer at home to one at their place of work. In order for the two machines to communicate both harmoniously and safely, however, measures need to be in place so as to ensure that the channel over which the devices interact isn’t going to be breached. That’s where SSH comes in.

In short, as one might guess from its name, SSH provides security. It acts as a shell to a given channel of data communication, effectively protecting it from would-be intruders. It achieves this aim chiefly through encryption of the data being transferred. Additionally, SSH employs a verification process, meaning it requires a coder to input their credentials before allowing them access to the foreign device. This way, a user can only commit changes to the system in question provided they are successfully authenticated and have the proper authorization.

# **Why is It Necessary?**

SSH doesn’t facilitate the actual act of communication between clients and servers- if a user’s end goal is simply to connect to and interact with a host with no regards for their data’s safety, the protocol isn’t necessary. That being said, most people and organizations need security; we want to know that what we exchange with a server is protected and that it isn’t going to fall into the wrong hands. This includes passwords, banking information, trade secrets, corporate documents, and so on.

Generally, however, data traverses the Internet in ‘plain text,’ meaning if the bits are intercepted, they’ll be plainly readable. This applies whether the information contained in a packet is confidential or not. As such, it is imperative that there should exist a means of communicating across unsecured networks securely- SSH is one such means. You can imagine SSH as a tunnel that connects the client to the host and shields the flow of data therein from the world outside. In reality, the packets being sent back and forth are still ‘visible.’ They can still be plucked or intercepted by external forces operating within the unsecured network at large- the difference is, they won’t be readable. SSH governs the encryption of the data such that even if it is accessed by an unauthorized third-party, they won’t be able to decrypt it.

# **The History of SSH: How Did it Come to Be?**

The original SSH, SSH-1, was created by a Finnish academic named Tatu Ylonen in the year 1995. After bearing witness to an attack on his school’s network, Ylonen was inspired to devise a more secure protocol for two devices communicating remotely than those that were available and in use at the time (such as rlogin). These popular protocols, among them TELNET and FTP, were not sufficiently ‘safe’, as many weren’t encrypted and thus, simply not as secure as they needed to be. So Ylonen set out to solve the problem, developing the first iteration of SSH and putting it up online at no cost to the end consumer.

SSH arrived at just the right time, filling the void its precursors could not. As the SSH userbase grew, so too did the need for more developers. By the end of 1995, SSH Communications Security was established. Over the years, as SSH gradually became bigger and better, the software changed, ultimately going closed-source. It wasn’t until 2006 that SSH-2 was invented, a protocol that significantly improved upon its predecessor’s design and was quickly adopted as the new standard.

Lastly, OSSH (created by Bjorn Gronvall) expanded upon an earlier, open source version of SSH-1, serving as a free alternative to the newer versions, that is until OpenSSH in turn took the OSSH source code and improved upon that. This evolutionary process resulted in the OpenSSH we know today, which remains the most widely utilized Secure Shell implementation. It can work with SSH-2, is still looked after, and is employed by a variety of operating systems.

# **How Does It Work?**

SSH uses the client-server model. An SSH server is specified by the SSH client and contacted accordingly. When using Putty, for instance, the first step is to provide the host’s name or IP address. The server replies and a safe, encrypted ‘pathway’ is established between the two devices; the Secure Shell Connection is established. The user must then sign in and provide the password pertaining to the account they have specified. Interestingly, the password isn’t displayed when typed, but is sent all the same. If this request is permitted and the verification is correct, the server responds by means of a terminal window.

SSH utilizes three encryption techniques – Symmetrical, Asymmetrical, and Hashing – to get the job done.

## Symmetric (or shared key) Encryption

This involves the use of a key to encrypt/decrypt the data being sent back and forth over the SSH channel. The client and the server, upon ‘meeting’ (that is, before the user has even authenticated themselves), create a key. This key is unique to the specific session and remains secret through the session’s lifetime, known solely to client and server. From its creation onwards, it is used to encrypt everything that is sent between the two machines. As password verification occurs after the channel has been established, these credentials are also encrypted and thus safe from prying eyes.

The client and server come by the secret token in question using a key exchange algorithm. Basically, having the key proper sent across the channel wouldn’t make sense, as it could be intercepted, so they encrypt it first (as explained in the Asymmetric Encryption section). Once they both have this same, hence symmetrical, key, data can afford to be passed back and forth. Essentially, the key allows the sending machine to encrypt and the receiving machine to decrypt. This way, data can afford to be made public as it travels across the larger network because it is already, effectively, locked.

Without the key, even if someone were to intercept a package, it would just be a jumble of unintelligible bits; they wouldn’t be able to unlock the message contained within, as they themselves don’t have the proper key to translate the data back into something readable. For added security, whenever the client and server connect, they have to agree upon a bidirectional cipher, one that is supported by both machines. Thus, the means of generating the key can differ depending on different client/server connections.

It’s important to stress that the algorithm that results in the symmetrical key’s creation leads to a temporary, variable key, meaning it’s only the key for the current session as long as said session continues. Ultimately, the secret key produced is then used in encrypting and decrypting all the data packets that get exchanged over the course of the session. Should the user log out and try to get back in again, a new key is established.

## Asymmetric Encryption

In contrast to the previous technology, asymmetric encryption uses two keys, or a pair- one public, the other private. In an SSH session, the only time this process is actually used is during the key exchange algorithm sequence. Establishing the secret key that is used in the symmetrical encryption that governs the bulk of a session requires its own process. In order for both the client and host to come about this secret key, they can’t simply send it over the unsecured network. That’s where asymmetric encryption comes in.

Both client and server create public-private key pairings. Simply put, the public key is used to encrypt data packets, and the private key is used to decrypt. The two keys are interrelated, but one cannot be derived from the other. The public keys are named as such because they are transferred between the machines before a secured connection is established, meaning they could potentially be intercepted. The client uses the public key to encrypt the data packets that it sends to the server and the server uses the corresponding private key to decrypt that data and vice versa. Remember, the public key cannot decrypt, thus a third-party can’t use it to access the encrypted stream.

The private keys, which do decrypt, are known solely to the relevant parties. By encrypting the local private key with the foreign public key, the private keys can actually be exchanged and still remain private. This works towards creating the symmetric key that is used throughout all further communications between the client and server. The server then tests the newly made symmetric key by sending the client a message and verifying if it responds appropriately.

## One-Way Hashing

Hashes are employed by SSH through HMACs (Hash-based Message Authentication Codes) as a means of ensuring messages received by the client are legitimate. Just as with the cipher selection, the client and server ‘decide upon’ an algorithm that will determine a distinct, fixed-length string for every command that is issued. This variable isn’t meant to be decrypted. Though it is derived from an input, it can’t be used in reverse to discern the data being sent. Instead, its purpose is that when the message is received, it too is hashed and the two values (the hash sent and the hash created based on the input that was ultimately received) are compared. If the hashes are the same in each case then it’s safe to assume that the message wasn’t altered unduly between being sent and being received. A MAC is sent after the symmetrically encrypted material so as to mark the end of the current packet.

The algorithm for determining the authentication codes is decided upon at the same time as the symmetric algorithm is.

The symmetrical key is derived using the Diffie-Hellman Key Exchange Algorithm. This is a complicated process in which a big prime number is manipulated through a series of operations to get a seed while a different prime is used for the private key. The public key is determined using the private key and another algorithm. The symmetric key is finally created by both the client and server using the local private key, the foreign public key, and the prime number. This should result in the same key being forged on either side (a key which third parties wouldn’t be able to determine). Overall, this only takes two to three round trips.

Once they’re inside the shell, the user can run administrative duties just as they normally would if the server was their physical machine. One can execute code and implement processes safely, trusting SSH to keep it safe.

# **Specifications**

According to the RFC (Request for Comments) of the IETF (Internet Engineering Task Force), the SSH Transport Layer protocol, every packet is of the form:

Packet\_length (uint32) – padding\_length (byte) – payload (bytes) – random\_padding(bytes) – mac

# **Additional Information**

Typically, SSH utilizes TCP port 22. The server listens for clients until one connects, whereupon the protocol comes into play. The two machines first set up the secure channel by which to communicate by determining the encryption methods. Servers will typically have different versions of the protocols so as to remain available to a wider array of clients. Provided the server boasts a protocol version that the client is using, the process can continue. Only after the safety measures are in place will the host accept the password input, using this to verify the user is who they say they are. Clients, in turn, are responsible for determining a symmetric connection has been established with the host. They also check that the host’s current credentials line up with those recorded.

Alternatively, users don’t have to use a password (in fact, some even recommend against it). It can actually be more secure to use key pairings, as in asymmetric encryption, to govern the authentication process.

# **To Reiterate: Why Is SSH Useful?**

As previously stated, SSH provides a secure means for a client to communicate with a remote server. Once the channel is up and running (and the client is signed in), the user can run all the same processes as they would be able to if they were directly connected to the server’s local network. This includes, among other things, the issuing and execution of code, the ability to run commands, and the secure transferring of files. The user can create, read, update, and delete files at will, the same as if these files were located on their physical machine.

Linux and macOS users have SSH readily available, while Windows programmers need to use a client, the most common of which (and the one Seneca students use to connect to the Matrix server) is Putty.

Overall, SSH exists to provide users with ease of mind. Trusting the shell to safeguard the communication channel allows administrators to rest assured that connecting to a remote server will be relatively safe. This surety allows people to do their jobs more easily. SSH is a proven defense against potential threats, effectively hampering data spillage and fending off unwanted interference.

Perhaps the most remarkable thing about the Secure Shell protocol is the amount of time it takes the machines involved to undergo the whole process and create the shell. In under a second – the blink of an eye – a client connects to a host, the public keys are exchanged, both a symmetrical connection and hash are established, and a newly secured communication channel exists where there wasn’t one. SSH’s only limitation isn’t the time it takes to undergo any of these individual encryption methods, but how long the bit streams spend in going back and forth across the network. Simply put, SSH works about as fast as one’s Internet connection does.

# **In Conclusion**

There’s a reason Telnet isn’t mentioned very much these days and it corresponds directly with the rise of SSH. The secure shell protocol is particularly good at what it does. Ultimately, connecting two machines across an unsecured network such as the Internet is unwise, as it means anyone with ill intent could potentially access one’s classified data. As such, it is necessary that a protocol such as SSH should exist. It is right to be cautious when transferring data over the Internet; SSH provides the means to do so easily, efficiently, and with peace of mind.

# **Bibliography**

Ylonen, T., Lonvick, C., SSH Communications Security Corp, Cisco Systems, Inc. (n.d.). The Secure Shell (SSH) Transport Layer Protocol. Retrieved November 23, 2019 from <https://tools.ietf.org/html/rfc4253>.

Ellingwood, J. (2014, October 22). Understanding the SSH Encryption and Connection Process. Retrieved November 25, 2019 from <https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process>.

Geerling, J. (2014, April 15). A brief history of SSH and remote access. Retrieved November 23, 2019 from <https://www.jeffgeerling.com/blog/brief-history-ssh-and-remote-access>.

Ladia, A. (2017, November 10). SSH Tutorial for Beginners - How Does SSH Work. Retrieved November 24, 2019 from <https://www.hostinger.com/tutorials/ssh-tutorial-how-does-ssh-work>.

Maheshwari, M. (2017, September 24). Understanding SSH workflow. Retrieved November 24, 2019 from <https://medium.com/@Magical_Mudit/understanding-ssh-workflow-66a0e8d4bf65>.

Rouse, M., Loshin, P., & Cobb, M. (n.d.). What is Secure Shell (SSH)? - Definition from WhatIs.com. Retrieved November 25, 2019 from <https://searchsecurity.techtarget.com/definition/Secure-Shell>.

Ssh.com (n.d.). SSH (Secure Shell) Home Page. Retrieved November 23, 2019 from <https://www.ssh.com/ssh>.

Wikipedia contributors. (n.d.). Secure Shell. Retrieved November 24, 2019 from <https://en.wikipedia.org/wiki/Secure_Shell>.