sshr: An SSH Proxy Server
Responsive to System Changes without Forcing Clients to Change

Hirofumi Tsuruta
SAKURA internet Inc.
Email: hi-tsuruta@sakura.ad.jp

Ryosuke Matsumoto
SAKURA internet Inc.
Email: r-matsumoto@sakura.ad.jp

Abstract—**To respond to various requests from users, web service infrastructure must change system configurations quickly and flexibly without making users aware of the system configuration. However, because SSH used as a secure remote connection service to a server must send a connection request by specifying the IP address or hostname of the server, the SSH client must know the changed information when the IP address or hostname is changed. To overcome this difficulty, a method exists by which a client tool such as gcloud command obtains the IP address or hostname of the destination server based on unique label information of each server. However, this method requires restrictions and changes to the tools used by the client side. Another method is to use a proxy server, such as SSH Piper, to obtain the IP address or hostname of the destination server based on the SSH username. In existing SSH proxy servers, the source code must be changed directly to change the proxy server behavior. As described herein, we propose an SSH proxy server which can follow system changes using hook functions that can be incorporated by system administrators without requiring restrictions or changes to the clients. The proposed method has high extensibility for system changes because the proxy server behavior can be changed easily merely by modifying the hook function to be incorporated. Furthermore, using the proposed method confirmed that the overhead of establishing an SSH session is about 20 ms, which is a short time during which the SSH client does not feel a delay when logging into the server with SSH.**

1. Introduction

As web services have become widely available in the world, their number of users is increasing. As a result, system administrators must change the configurations of the managed systems quickly and flexibly according to the situation to respond to various requests from service users [4]. For example, when the number of accesses to web services suddenly increases or decreases, the system administrator responds to such changes by scale-out or scale-in servers [12] to reduce opportunity loss and operation costs of the services. Furthermore, with the expansion of web services, it is also necessary to introduce new servers with new roles [1]. It is necessary for service growth to change the system configuration quickly and flexibly in response to changes in the service environment.

For situations in which the system configuration must be changed quickly in response to various demands for services, the operation and management of the system must also be capable of accommodating the change. However, Secure Shell (SSH) [14], which is used widely as a secure remote connection service to servers, requires SSH clients to specify a server's IP address or hostname to send a connection request. Therefore, if the IP address or hostname of the server is changed, then the administrator must inform each client of the changed information. Each client must therefore follow their changes.

Currently, several methods allow clients to connect to the server with SSH without being aware of the IP address and hostname of the destination server. One is a method by which the client tool obtains the IP address or hostname of the destination server based on the unique label information for each server. An example is the gcloud command [6] of Google Cloud Platform (GCP). Using this method, the SSH client can connect to the target server transparently using only the label information associated with the server without necessitating awareness of the system configuration and its changes. However, this approach forces SSH clients to use special client tools such as the gcloud command. Furthermore, when a change occurs in the client tool, all clients must follow the change by upgrading the tool.

Another method is one by which a proxy server located between the client and the server obtains the IP address or hostname of the connection destination based on information of the SSH request. For existing SSH proxy servers, open source software called SSH Piper [2] exists. By connecting with SSH via SSH Piper, the SSH client is not forced to use special client tools. Furthermore, the SSH client can connect to the target server associated with the SSH username without having to be aware of the server's IP address, hostname, or its change. The issue of SSH Piper is that the system administrator cannot freely incorporate or change the logic for determining the connection server based on the username. If the administrator wants to change the logic for determining the connection server, then the administrator must make changes directly to the source code of SSH Piper. Therefore, the extensibility to system changes is not high.

As described in this paper, we proposed an SSH proxy server which can follow system changes using hook functions that can be incorporated freely by system administrators without requiring either restriction or change of the client tools used by SSH clients. The proposed SSH proxy server was designated as sshr [8]. The sshr allows system administrators to implement and incorporate hook functions for determining the connection server freely based on the SSH username. As a result, even if the IP address or hostname of the destination server changes, the administrator need not notify each client of the changed information. The client always connects to the target server associated with the username. Moreover, sshr has functions to extend the user authentication required for establishing an SSH session. For example, when sshr receives a request for public key authentication from a client, the process used to seek the client's public key can be incorporated into sshr as a hook function. This feature allows the system administrator to manage the client's public key in a free data format such as a database. In this way, sshr can extend user authentication of SSH protocol programmatically using hook functions.

This paper is configured as described below. Section 2 presents related works and their issues. Section 3 describes the proposal given herein. In section 4, a use case with the proposed method is described with specific examples. Section 5 presents results of performance evaluation of the proposed method. Finally, we summarize our contributions in section 6.

2. Related Works

For situations in which the system configuration must be changed quickly in response to various requests for services, the operation and management of the system must be able to accommodate the change. By contrast, for SSH, if a change occurs in the server IP address or hostname, the administrator must notify each client of the changed information. The client must then acknowledge and accommodate the change. This section presents a summary of existing methods to solve this SSH issue and describes their features and problems.

2.1. SSH client tool

A command line tool for managing computing resources on GCP is gcloud. By specifying the instance name with the "gcloud compute ssh" command [7], the client can connect to the target server by SSH. Actually, the "gcloud compute ssh" command is a wrapper for the ssh(1) command [15]. It has the function of obtaining the IP address from the configuration information managed by GCP using the instance name as a key before sending an SSH request. Consequently, if the SSH client knows the unique instance name for each server, then the SSH client can connect transparently with SSH without being aware of the IP address of the target server or its change.

Another method is to use Consul [9], a cluster management tool developed by HashiCorp. Consul can manage members in the cluster and an HTTP API to refer to the member information. Using these functions, one can obtain the IP address of a target server based on some tag information defined by the administrator. Therefore, one can create a client tool such as consult [11], which sends an SSH request using the IP address obtained from the Consul HTTP API.

In both methods, the SSH clients can connect to the target server using the label information provided for the respective servers. Even if the IP address of the target server is changed, the client tool can play the role of following the change without the client being aware of the change. However, these methods force the SSH clients to use special SSH client tools such as the gcloud command. Therefore, if the program of client tool must be changed in accordance with the change on the system side, then all clients using the tool are forced to make changes such as upgrading the version of the tool.

2.2. SSH Proxy server

SSH Piper [2] is an SSH proxy server developed as open source software on GitHub. A salient feature of SSH Piper is that, when receiving an SSH request, the connection destination server can be determined from the SSH username by SSH Piper. When using SSH Piper for SSH connection, if the SSH client knows only the IP address or hostname of the proxy server using the SSH Piper, then the SSH client can connect to the target server without being aware of the server's IP address or hostname, or related changes.

With SSH Piper, the role of determining the connection destination is not assigned to the client tool but to the proxy server. For that reason, no need exists to restrict the client tools used by the SSH clients. Nevertheless, the system administrator cannot freely implement and incorporate the logic to determine the destination server. As a result, it is not possible to design the data format freely to manage link information between the SSH client and the SSH server. For example, if the administrator wants to manage the link information between the SSH client and the SSH server in the database, then it would be necessary to prepare a table schema defined by the SSH Piper. If the administrators want to extend the logic for determining the connection destination from the existing logic, then they must make changes directly to the source code of the SSH Piper.

Another method to build an SSH proxy server that can determine the connection destination such as SSH Piper is to extend sshd [16], which is currently the most widely used SSH server daemon program. Compared to implementing a proxy server from scratch, the advantage of extending sshd is that developers need not implement authentication processes because they can reuse the highly reliable authentication processes of sshd. However, because sshd does not support a feature extension mechanism, developers must make direct changes to the sshd source code to extend that feature.

In neither method does the need exist to restrict the SSH client tools. The SSH client can connect to the target server associated with the SSH username. Even if the IP address

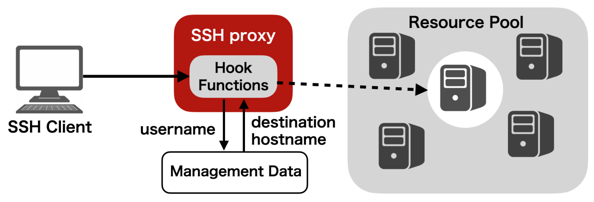

Figure 1. Schematic diagram of the proposed method.

of the target server is changed, the proxy server can play the role of following the change without the clients being aware of the change. One issue is that existing proxy servers do not support the mechanism for function extension of proxy servers. Therefore, the extensibility to system changes is not high.

3. Proposed Method

As described in section 2.1, when the client tool plays the role of accommodating the system change, the issue is to restrict the client tool used by the SSH client. However, as described in section 2.2, when the proxy server plays the role of following the change, it requires no SSH client to restrict the client tools. In fact, the issue is that the extendability of the proxy server for the system change is not high. To overcome these difficulties, we propose an SSH proxy server that can follow the system changes using hook functions that can be incorporated freely by the administrator without requiring the client to limit or change the client tools used.

3.1. Architecture overview

In the proposed method, we adopted an architecture that allows free incorporation by the system administrator of the processing executed during the SSH request as hook functions. Figure 1 is a schematic diagram of the proposed method. The SSH proxy server receives the connection request from the client and determines the destination server. Results show that if the SSH client only knows the IP address or hostname of the proxy server, then the client need not be aware of the configuration of the system behind the proxy server and its changes. From the viewpoint of the system administrator, if the IP address or hostname of the proxy server already provided to the client does not change, then the system configuration can be changed freely without notifying the client. Therefore, it is practical to assign a virtual IP address to the load balancer placed in front of the proxy server so that all SSH connections from clients are accepted with the same IP address. Results show that, even if the IP address of the proxy server changes, or if the administrators want to scale out the proxy server, they need not request any change for the SSH clients.

When connecting to a server with SSH, one must perform user authentication [13] to ascertain whether the client has authority to use the target server. The proposed method supports password authentication and public key authentication as authentication methods. When user authentication is performed through a proxy server, how the proxy server mediates authentication between the client and server differs depending on the authentication method. For password authentication, the proxy server can send the request packet to the SSH server without interpreting the authentication request received from the client. Authentication is performed on the SSH server. By contrast, for public key authentication, the proxy server cannot delegate authentication to the SSH server. This is true because an SSH session has a unique identifier for each session called a session ID [13]. The data including the session ID are encrypted by a private key to send an authentication request. The proxy server has two SSH sessions between the client and the proxy server and between the proxy server and the SSH server. As one might expect, each has a different session ID. Therefore, when performing public key authentication via a proxy server, one must perform authentication in two steps between the client and the proxy server and between the proxy server and the SSH server.

With the proposed method, multiple hook functions with a specific role can be incorporated into the SSH proxy server. These functions are executed sequentially in the process of processing the SSH request. As a hook function that can be embedded for the proxy server, a function exists that obtains the IP address or hostname of the connection destination server based on the SSH username. Additionally, it has a hook function to extend public key authentication between the determined connection destination server and client. Specifically, a function exists to search the client's public key used for the authentication between the client and the proxy server. Also, a function exists to search the private key used for the authentication between the proxy server and the SSH server. Using these, the connection destination determination and public key authentication in the SSH proxy server can be programmably extended. Furthermore, the system administrator can manage the link information between the client and the server and the public key used for authentication in a free data format such as a database.

3.2. Implementation

We developed an SSH proxy server called sshr [8] using Go [5] to realize the proposed architecture. The proxy server behavior can be extended by incorporating the function implemented by the system administrator in Go into sshr. Actually, sshr supports the following three hook functions.

(1) FindUpstreamHook
(2) FetchAuthorizedKeysHook
(3) FetchPrivateKeyHook

(1) is a function to obtain the IP address or hostname of the destination server based on the SSH username. (2) and (3) are hook functions required when using public key authentication. (2) is a function to retrieve the client's public key. (3) is a function to retrieve the private key used to authenticate to the destination server from the sshr server. Details of the usage of (2) and (3) are described later.

An important issue in implementing an SSH proxy server is how to perform authentication between the client and server via the proxy server. As described in section 3.1, when public key authentication is used, authentication must be performed in two steps between the client and the sshr server and between the sshr server and the SSH server. First, between the client and the sshr server, the sshr server has a FetchAuthorizedKeysHook function to search for the client's public key. This function is equivalent to AuthorizedKeysCommand of sshd. Using this, the system administrator can centrally manage the client's public key in a free data format such as a database. Next, between the sshr server and the SSH server, the sshr server has no client private key. Therefore, it is necessary to use another private key for sending the authentication request to the SSH server. The first authentication step between the client and the sshr server verifies that the client is allowed to connect to the target server. Therefore, in the second authentication step between the sshr server and the SSH server, the SSH server permits all authentication requests using the specific private keys having the sshr server. At this time, FetchPrivateKey-Hook is provided as a hook function to seek the private key used by the sshr server to authenticate to the SSH server.

Next, the procedures for sshr after successfully authenticating and establishing the SSH session are described. The hook functions embedded into sshr are all executed before the SSH session is established. After session establishment, sshr does not interpret the packets received from the client and server. It only forwards the packets in both directions. For this reason, it is assumed that the overhead because of the execution of the hook function occurs when the SSH session establishment. This point will be evaluated in section 5.

3.3. Security risks

The security risks of unauthorized access when using sshr are discussed here. When password authentication is used as the user authentication method, the sshr server forwards the packet to the SSH server without reading the password. In other words, only the SSH server knows the client password and can authenticate. For that reason, using the sshr server does not affect the risk of breaking through the authentication.

Public key authentication requires two-step authentication for each session between the client and the proxy server and between the proxy server and the SSH server. As described in section 3.2, the SSH server accepts all requests using the specific private keys of the sshr server. For this reason, if one assumes that the security problem of the sshr server or the vulnerability of the sshr daemon exists, then the private key of the sshr server might be leaked, or any command using the private key from a malicious user might be executed.

To mitigate these risks, countermeasures are described for each of the sshr server and the SSH server. First, in the sshr server, the security of the server must be strengthened to prevent leakage of the private key that it holds. For example, it is effective to minimize the number of users who can access the sshr server and limit the IP address range that is accessible only to the networks within the organization. Next, to reduce the damage that occurs when a malicious user executes an arbitrary command via sshr, it is useful to run the sshr daemon as a user without root privileges. This countermeasure is effective in restricting the operation on the sshr server when an arbitrary command is executed. However, the command is executed with the same authority as the sshr daemon. Therefore, operation on the SSH server is possible.

Next, as a countermeasure for the SSH server, remote login as root user must not be allowed on the SSH server. For example, with sshd, one can restrict remote login as root user using PermitRootLogin of sshd_config. This requirement can limit the authority of operations on the SSH server. It is also effective to restrict the connection source IP in case the private key of the sshr server leaks to the outside. For example, in the authorized_keys file [16] of sshd, the connection source IP can be restricted for each public key [3]. Therefore, configuring the SSH server to allow connection only from the IP address of the sshr server is effective. Results show that it is possible to prevent damage when the private key of the sshr server leaks to the outside.

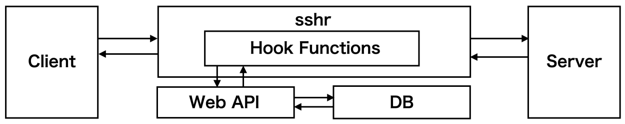

Figure 2. Example of system configuration.

4. Use case

Figure 2 presents an example of a system configuration which adopts the proposed method. Here, the explanation for each role in Figure 2 is described. Because the Client role has no restrictions on the client tools, ssh(1) command [15] or a GUI-based SSH client is assumed to be used. In addition, because no restrictions exist on the tools used for the Server role, sshd [16] is assumed. The DB role is a database server that manages the usernames of SSH clients and the SSH servers. Using the proposed method, the client can connect to the server associated with the username. Therefore, the database manages the association between the username and the hostname of the server. Furthermore, manage the client's public key in a database is useful to authenticate the client's public key on the sshr server. One can refer to information stored in the database directly from sshr, but it is also possible to refer to the information via Web API. In this way, because sshr employs an architecture that can extend the proxy server behavior programmatically as hook functions, the method of acquiring information from the database can be selected freely.

Because the sshr role has three executable hook functions, the processing assumed for each is explained. First, FindUpstreamHook used to determine the connection destination obtains information of the destination server from the Web API as an HTTP client. Next, FetchAuthorizedKeysHook, which is used to search for the client's public key, performs processing to obtain the client's public key stored in the database via the Web API as an HTTP client. Finally, FetchPrivateKeyHook, which searches for the private key used to authenticate to the connection destination server from the sshr server, is presumed to refer to the private key stored in the local storage of the sshr server. This is true because managing private keys in a database and obtaining them via a network increases the risk of leakage.

When the system described above is constructed, following the change of a system can be described with examples. The following are examples of possible system changes.

(1) Change the server used by the specific client
(2) Change the table configuration in the database
(3) Change the Web API specifications

For (1), merely updating the information in the database is sufficient. Because sshr refers to the association information between the client and the server from the database at each request, one can switch the connection destination of the client merely by updating the information stored in the database. For (2), it is sufficient to change the processing of Web API in accordance with the change on table configuration; sshr need not be changed. For (3), changing the hook function of sshr is necessary. The system administrator must modify the hook function according to the change of Web API specifications and deploy the executable binary file. As presented in the examples above, the proposed method has high extensibility to system changes because the proxy server itself need not be changed or the behavior can be controlled merely by modifying the hook function for various possible system changes.

5. Evaluation

In the proposed method, the processing of connection destination determination and user authentication can be extended programmably. Actually, one can follow the system changes without making the client aware of the change. However, the hook function executed when the session establishment causes communication overhead. By measuring this overhead, one can evaluate the effects of the proposed sshr on the processing time of the SSH connection and discuss whether sshr is useful in a practical environment.

The overhead is measured when the uname command is executed via sshr and when the file is transferred by scp (secure copy) via sshr. Only public key authentication is used as the authentication method between the client and the server. In the case of password authentication, sshr forwards the packet to the SSH server without interpreting the authentication request received from the client. By contrast, for public key authentication, a hook function is executed to search for the client's public key. Because of this difference, public key authentication takes a longer time to authenticate than password authentication. For that reason, evaluation is performed under the condition that the overhead lengthens.

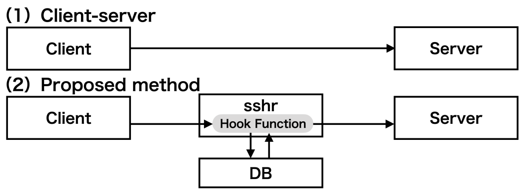

Figure 3. Schematic diagram of the experiment environment.

TABLE 1. EXPERIMENT ENVIRONMENT

Role	Item	Specifications
Client	CPU	Intel Xeon CPU E5-2650 v3 2.30 GHz 1 core
	Memory	1 GBytes
	SSH Client	OpenSSH 7.9
Server	CPU	Intel Xeon CPU E5-2650 v3 2.30 GHz 1 core
	Memory	1 GBytes
	SSH Server	OpenSSH 7.9
sshr	CPU	Intel Xeon CPU E5-2650 v3 2.30 GHz 1 core
	Memory	1 GBytes
	sshr	sshr v0.1.6
DB	CPU	Intel Xeon CPU E5-2650 v3 2.30 GHz 1 core
	Memory	1 GBytes
	Database	MySQL 5.7.27

Figure 3 portrays a schematic diagram of the experimental environment. The experiment compares the normal SSH connection between the client and server without sshr and the SSH connection via sshr. Table 1 presents the specifications and software versions of the respective roles. The operating system (OS) of each role is CentOS 7.6.1810 Kernel 3.10.0. Each measurement of the execution time was performed 100 times. The average value was calculated. The hook functions executed by sshr refer to a remote database server in both the connection destination determination process (FindUpstreamHook) and the client's public key search process (FetchAuthorizedKeysHook). The database includes no records other than the target record.

Table 2 presents measurement results of executing the uname command. The overhead of executing the uname command was 22 ms when sshr mediated the SSH connection between the client and server. This time is sufficiently short that the SSH client senses no delay [10] when establishing the SSH session. Therefore, assuming system management in which multiple clients log in to a specific server and perform server operations, the overhead of sshr is sufficiently small to use in a practical environment.

Figure 4 presents the relation between the file size transferred by scp and the time taken for file transfer. When transferring a 20 MB file, the overhead caused by sshr was 48 ms. This result indicates that the overhead caused by sshr includes effects other than the hook function because the overhead is larger than the result obtained by executing the uname command. Furthermore, results demonstrate that

TABLE 2. MEASUREMENT RESULT OF UNAME COMMAND EXECUTION

Environment	Execution time/ms (100 times average)
Client-server	448
Proposed method	470

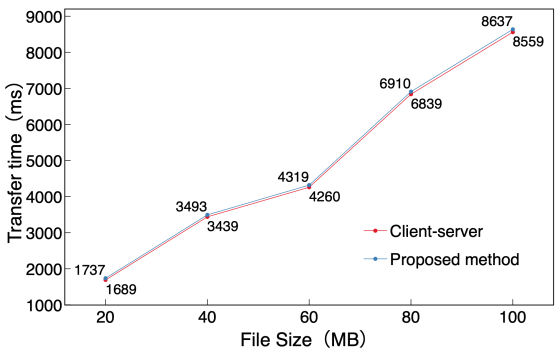

Figure 4. Relation between the file size transferred by scp and the transfer time.

the overhead increases as the file size increases. This result is probably attributable to the overhead caused by the sshr server forwarding the packet compared to sending the packet directly from the client to the SSH server. However, when particularly addressing the case of 100 MB file transfer, the increased rate of transfer time is only about 0.9%. Moreover, the ratio of overhead caused by sshr to the total transfer time is small. Therefore, except for the case of transferring numerous files via sshr, the overhead caused by sshr is regarded as practically negligible.

With the public key authentication method, the sshr server acquires the connection destination server and the public key from the management data. The acquisition time strongly affects the overhead. Therefore, if the response time of a database or Web API is long, then the time required for establishing an SSH session via sshr will increase accordingly. In such cases, one must take measures to shorten the response time.

6. Conclusion

In this paper, we proposed an SSH proxy server called sshr, which can follow system changes using hook functions that can be incorporated freely by system administrators without requiring either restriction or change of the client tools used by SSH clients. The system administrator can freely implement and introduce a hook function for sshr to ascertain the connection destination server from the SSH username. Consequently, even if the IP address or hostname of the destination server changes, the administrator need not notify each client of the changed information. Moreover, the client need not follow the change. Furthermore, sshr employs an architecture that can programmably extend the user authentication of the SSH protocol, such as incorporating a hook function to search for the client's public key. Therefore, the administrator can manage the client's public key in a free data format. Furthermore, experiments show that the overhead of establishing an SSH session via sshr is about 20 ms, which is a sufficiently short time that clients do not sense a delay [10] when logging into the server.

Future studies will evaluate the usefulness of sshr through the construction of a practical system to demonstrate its benefits. Using the hook function, one can select the most suitable destination server according to the surrounding situation. For example, it is possible to provide a server with the lowest load by analyzing the load status of the servers using machine learning. Next, the current sshr has a limitation that one SSH username is associated with only one server. To overcome this limitation, we plan to introduce a mechanism in which sshr returns a list of servers associated with the SSH username to the client, and the client can interactively select a connection destination from the list.

Acknowledgments

We express our sincere gratitude to members of Hosting Department, GMO Pepabo, Inc. for their great support and advice in developing the sshr proposed in this paper.

References

[1] B. Furht and A. Escalante, *Handbook of Cloud Computing*. Springer, 2010, ch. Service Scalability Over the Cloud.

[2] Boshi Lian, *SSH Piper*, https://github.com/tg123/sshpiper/.

[3] D. J. Barret, R. E. Silverman, R. G. Byrnes, *SSH, The Secure Shell: The Definitive Guide*, 2nd. O'Reilly Media, Inc., 2005, ch. A Recommended Setup.

[4] G. Galante and L. C. E. d. Bona, "A Survey on Cloud Computing Elasticity," in *2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 2012, pp. 263–270.

[5] Golang.org, *The Go Programming Language*, http://golang.org.

[6] Google Cloud Platform, *gcloud command-line tool overview*, https://cloud.google.com/sdk/gcloud/.

[7] Google Cloud Platform, *gcloud compute ssh*, https://cloud.google.com/sdk/gcloud/reference/compute/ssh.

[8] H. Tsuruta, *sshr*, https://github.com/tsurubee/sshr.

[9] HashiCorp, *consul*, https://github.com/hashicorp/consul.

[10] J. Nielsen, *Usability Engineering*. Morgan Kaufmann Publishers, 1993, ch. Response Times: The 3 Important Limits.

[11] Outbrain, *Consult*, https://github.com/outbrain/consult.

[12] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[13] T. Ylonen, *The Secure Shell (SSH) Authentication Protocol*, RFC 4252, 2006.

[14] T. Ylonen, *The Secure Shell (SSH) Protocol Architecture*, RFC 4251, 2006.

[15] The OpenBSD Foundation, *ssh – OpenSSH SSH client*, https://man.openbsd.org/ssh.1.

[16] The OpenBSD Foundation, *sshd – OpenSSH SSH daemon*, https://man.openbsd.org/sshd.8.