Understand Secure Shell Packet Exchange

Contents

Introduction

This document describes packet level exchange during Secure Shell (SSH) negotiation.

Prerequisites

Requirements

Cisco recommends that you have knowledge of basic security concepts:

- Authentication
- Confidentiality
- Integrity
- Key Exchange Methods

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

SSH Protocol

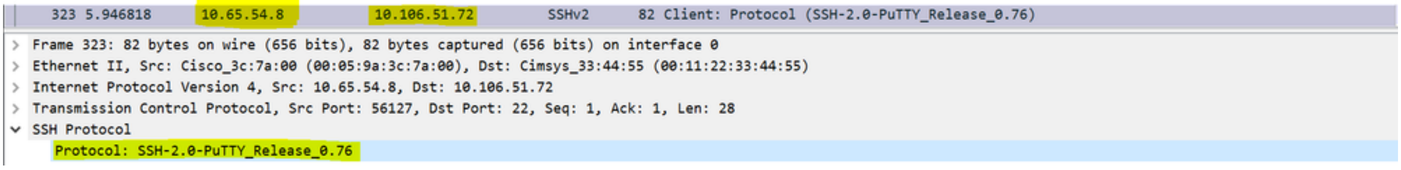
The SSH protocol is a method for secure remote log in from one computer to another. SSH applications are based on a client-server architecture, connecting an SSH client instance with an SSH server.

SSH Exchange

1. The first step of SSH is called Identification String Exchange.

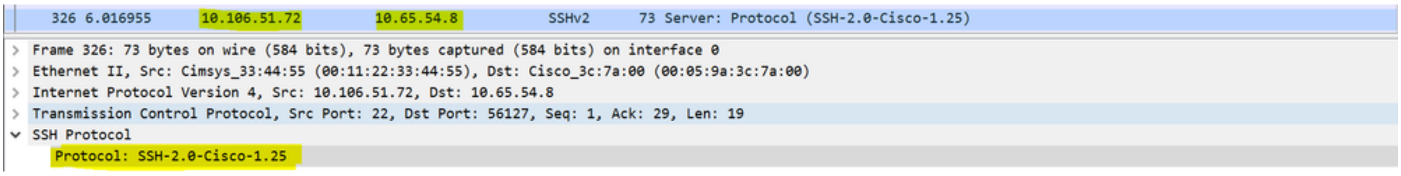
1.1. The client constructs a packet and sends it to the server containing:

- SSH-Protocol Version
- Software Version



Client Protocol Version and Software Version

1.2. The server responds with its own Identification String Exchange, including its SSH protocol version and software version.

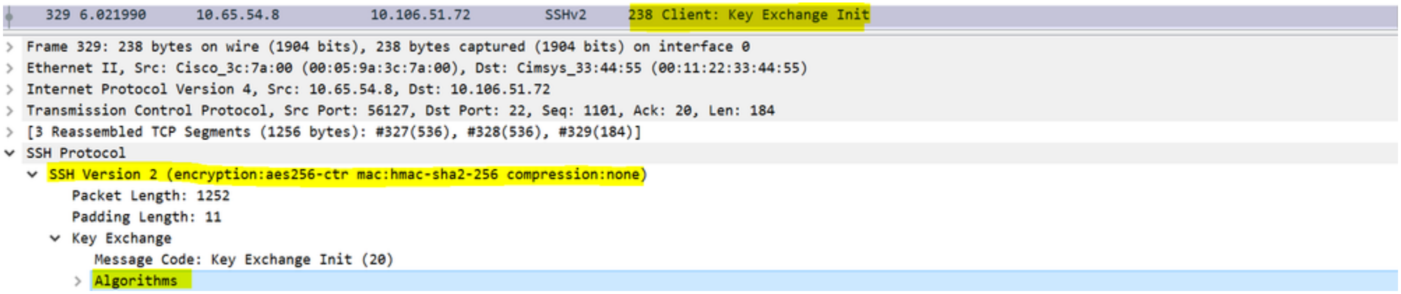


Server Protocol Version and Software Version

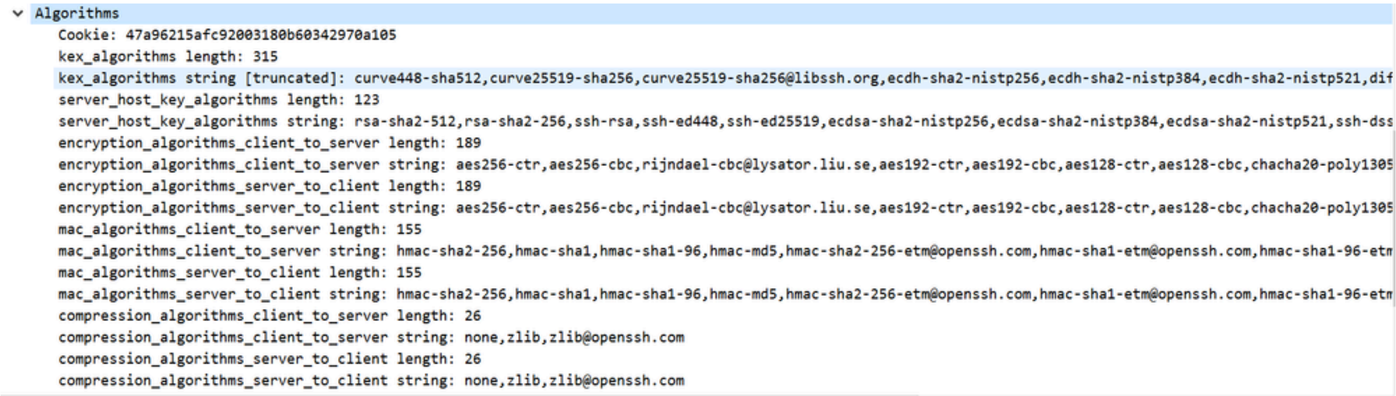
2. Next Step is Algorithm Negotiation. In this step, both Client and Server negotiate these algorithms:

- Keyexchange
- Encryption
- Hash-based Message Authentication Code (HMAC)
- Compression

2.1. The client sends a Key Exchange Init message to the server, specifying the algorithms it supports. The algorithms are listed in order of preference.



Client Key Exchange Init



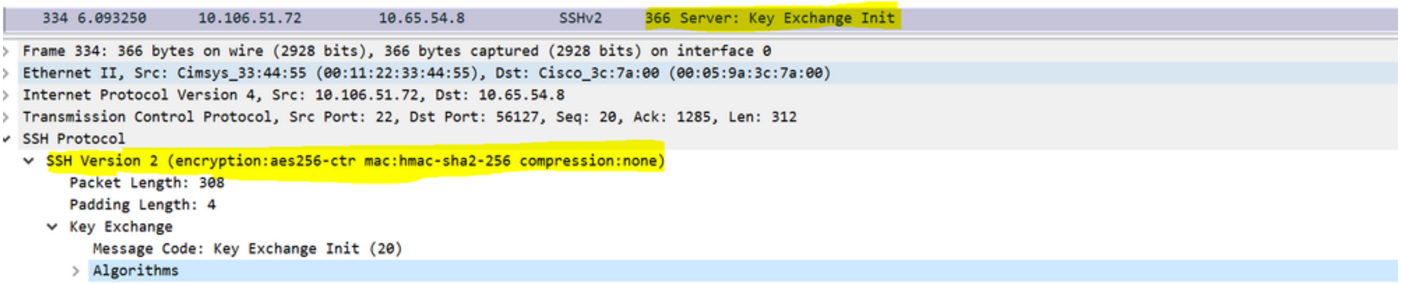
```
Algorithms
    Cookie: 47a96215afc92003180b60342970a105
    kex_algorithms length: 315
    kex_algorithms string [truncated]: curve448-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,dif
    server_host_key_algorithms length: 123
    server_host_key_algorithms string: rsa-sha2-512,rsa-sha2-256,ssh-rsa,ssh-ed448,ssh-ed25519,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-dss
    encryption_algorithms_client_to_server length: 189
    encryption_algorithms_client_to_server string: aes256-ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-ctr,aes128-cbc,chacha20-poly1305
    encryption_algorithms_server_to_client length: 189
    encryption_algorithms_server_to_client string: aes256-ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-ctr,aes128-cbc,chacha20-poly1305
    mac_algorithms_client_to_server length: 155
    mac_algorithms_client_to_server string: hmac-sha2-256,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-sha2-256-etm@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha1-96-etm
    mac_algorithms_server_to_client length: 155
    mac_algorithms_server_to_client string: hmac-sha2-256,hmac-sha1,hmac-sha1-96,hmac-md5,hmac-sha2-256-etm@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha1-96-etm
    compression_algorithms_client_to_server length: 26
    compression_algorithms_client_to_server string: none,zlib,zlib@openssh.com
    compression_algorithms_server_to_client length: 26
    compression_algorithms_server_to_client string: none,zlib,zlib@openssh.com
```

Client Supported Algorithms

2.2. The server responds with its own Key Exchange Init message, listing the algorithms it supports.

2.3. Since these messages are exchanged concurrently, both parties compare their algorithm lists. If there is a match in the algorithms supported by both sides, they proceed to the next step. If there is no exact match, the server selects the first algorithm from the client's list that it also supports.

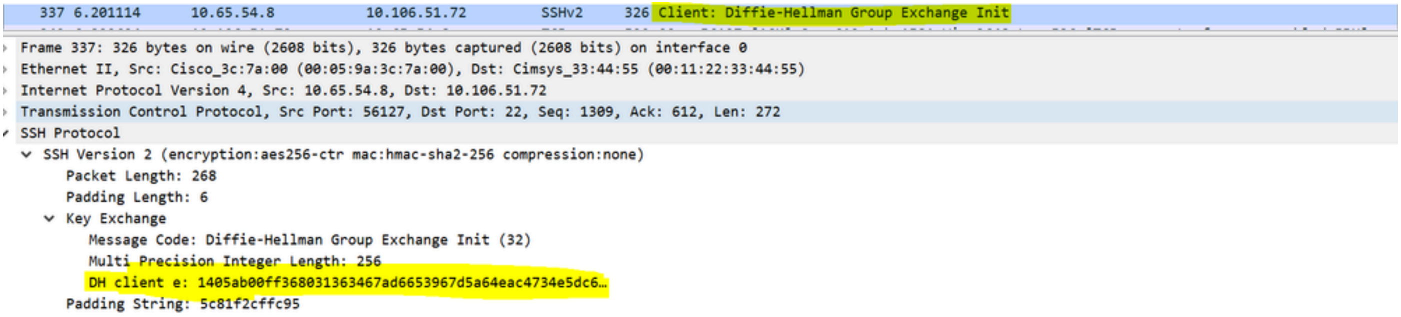
Note: If the client and server cannot agree on a common algorithm, the key exchange fails.



Server Key Exchange Init

3. Next, both sides enter the Key Exchange phase to generate shared secret using DH key exchange and authenticate the server:

3.1. The client generates a keypair, Public and Private, and sends the DH Public key in the Diffie-Hellman Group Exchange Init packet. This key pair is used for secret key calculation.



Client Diffie-Hellman Group Exchange Init

3.2. The server generates its own Public and Private key pair. It uses the client's public key and its own key pair to compute the shared secret.

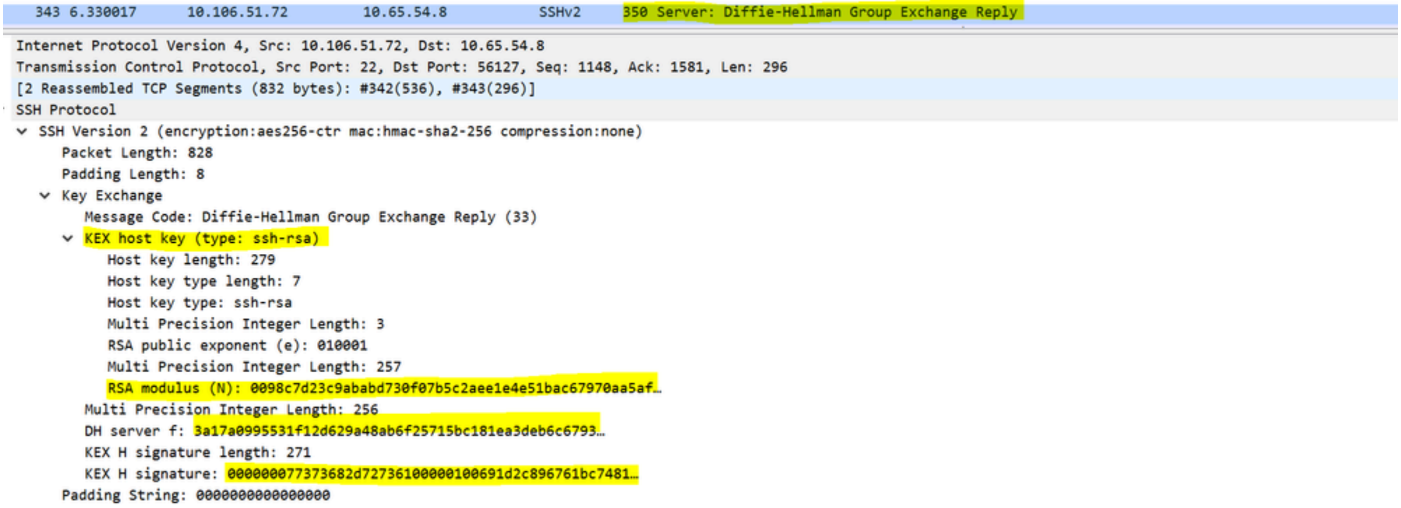
3.3. The Server also computes an Exchange Hash with these inputs:

- Client Identification String
- Server Identification String
- Payload of Client Key Exchange Init
- Payload of Server Key Exchange Init
- Server Public-key from Host keys (RSA key pair)
- Client DH Public Key
- Server DH Public Key
- Shared Secret Key

3.4. After computing hash, server signs it with its RSA Private Key.

3.5. The Server constructs a message Diffie-Hellman Group Exchange that includes:

- RSA Public Key of Sever (to help the client authenticate the server)
- DH Public key of Server (for calculating the shared secret)
- HASH (to authenticate the server and prove that the server has generated the shared secret, as the secret key is part of the hash computation)

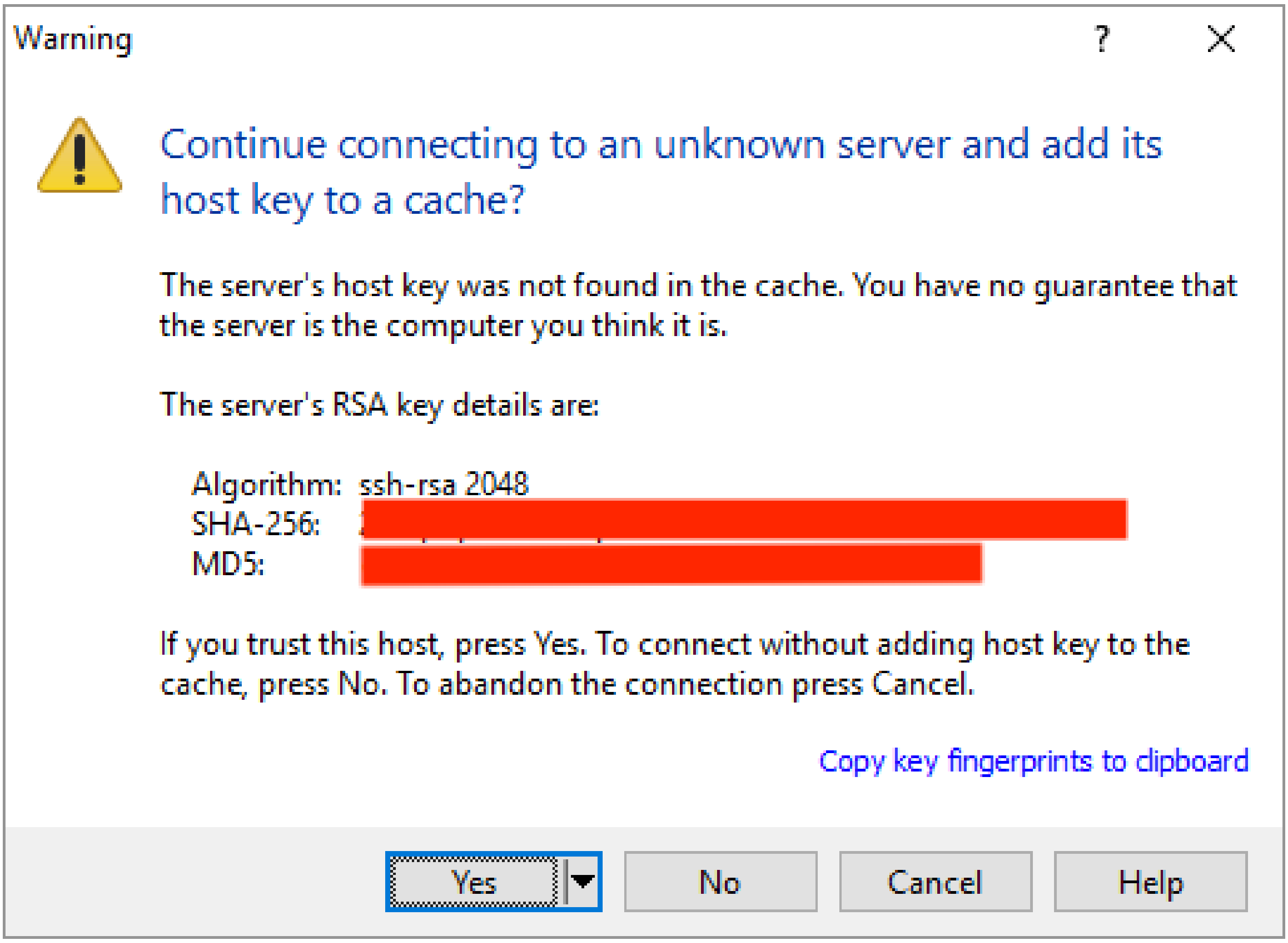


Server Diffie-Hellman Group Exchange Reply

3.6. After receiving the Diffie-Hellman Group Exchange Reply, the client computes the hash in the same way and compares it with the received hash, decrypting it using the server's RSA Public Key.

3.7. Before decrypting the received HASH, the client must verify the server's public key. This verification is done through a digital certificate signed by a Certificate Authority (CA). If the certificate does not exist, it is up to the client to decide whether to accept the server's public key.

Note: When you use SSH to log into a device for the first time that does not use a digital certificate, you can encounter a pop-up asking you to manually accept the server's public key. To avoid seeing this pop-up every time you connect, you can choose to add the server's host key to your cache.

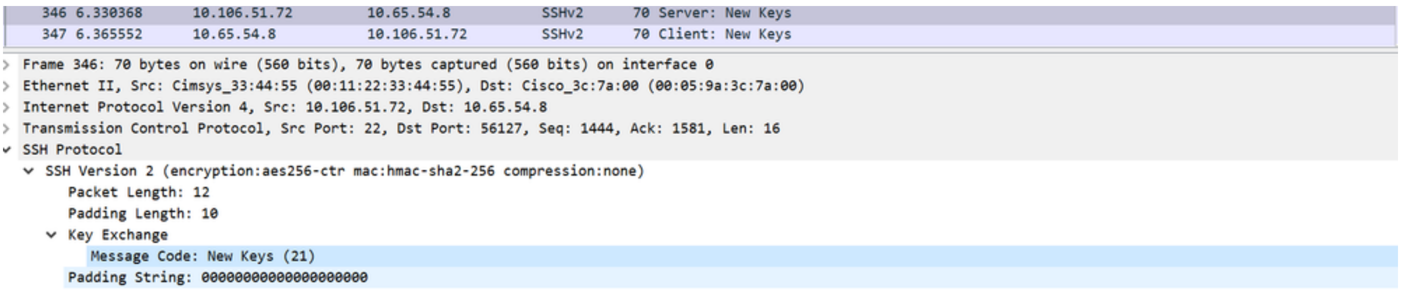


Server Public Key

4. Since the Shared secret is now generated, both ends use it to derive these keys:

- Encryption keys
- IV Keys - These are random numbers used as input to symmetrical algorithms to enhance security.
- Integrity keys

The end of the key exchange is signaled by the exchange of the NEW KEYS message, which informs each party that all future messages are encrypted and protected using these new keys.



Client and Server New Keys

5. The final step is the Service Request. The client sends an SSH Service Request packet to the server to initiate user authentication. The server responds with an SSH Service Accept message, prompting the client to log in. This exchange occurs over the established secure channel.

Related Information

- Configure SSH on Routers and Switches
- The Secure Shell (SSH) Transport Layer Protocol
- Cisco Technical Support & Downloads