

Understand Secure Shell Packet Exchange

Contents

[Introduction](#)

[Prerequisites](#)

[Requirements](#)

[Components Used](#)

[SSH Protocol](#)

[SSH Exchange](#)

[Related Information](#)

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

323	5.946818	10.65.54.8	10.106.51.72	SSHv2	82 Client: Protocol (SSH-2.0-PuTTY_Release_0.76)
> Frame 323: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0					
> Ethernet II, Src: Cisco_3c:7a:00 (00:05:9a:3c:7a:00), Dst: Cimsys_33:44:55 (00:11:22:33:44:55)					
> Internet Protocol Version 4, Src: 10.65.54.8, Dst: 10.106.51.72					
> Transmission Control Protocol, Src Port: 56127, Dst Port: 22, Seq: 1, Ack: 1, Len: 28					
SSH Protocol					
Protocol: SSH-2.0-PuTTY_Release_0.76					

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.

326	6.016955	10.106.51.72	10.65.54.8	SSHv2	73 Server: Protocol (SSH-2.0-Cisco-1.25)
> Frame 326: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0					
> Ethernet II, Src: Cimsys_33:44:55 (00:11:22:33:44:55), Dst: Cisco_3c:7a:00 (00:05:9a:3c:7a:00)					
> Internet Protocol Version 4, Src: 10.106.51.72, Dst: 10.65.54.8					
> Transmission Control Protocol, Src Port: 22, Dst Port: 56127, Seq: 1, Ack: 29, Len: 19					
SSH Protocol					
Protocol: SSH-2.0-Cisco-1.25					

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.

329	6.021990	10.65.54.8	10.106.51.72	SSHv2	238 Client: Key Exchange Init
> Frame 329: 238 bytes on wire (1904 bits), 238 bytes captured (1904 bits) on interface 0					
> Ethernet II, Src: Cisco_3c:7a:00 (00:05:9a:3c:7a:00), Dst: Cimsys_33:44:55 (00:11:22:33:44:55)					
> Internet Protocol Version 4, Src: 10.65.54.8, Dst: 10.106.51.72					
> Transmission Control Protocol, Src Port: 56127, Dst Port: 22, Seq: 1101, Ack: 20, Len: 184					
> [3 Reassembled TCP Segments (1256 bytes): #327(536), #328(536), #329(184)]					
SSH Protocol					
SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)					
Packet Length: 1252					
Padding Length: 11					
Key Exchange					
Message Code: Key Exchange Init (20)					
Algorithms					

Client Key Exchange Init

```

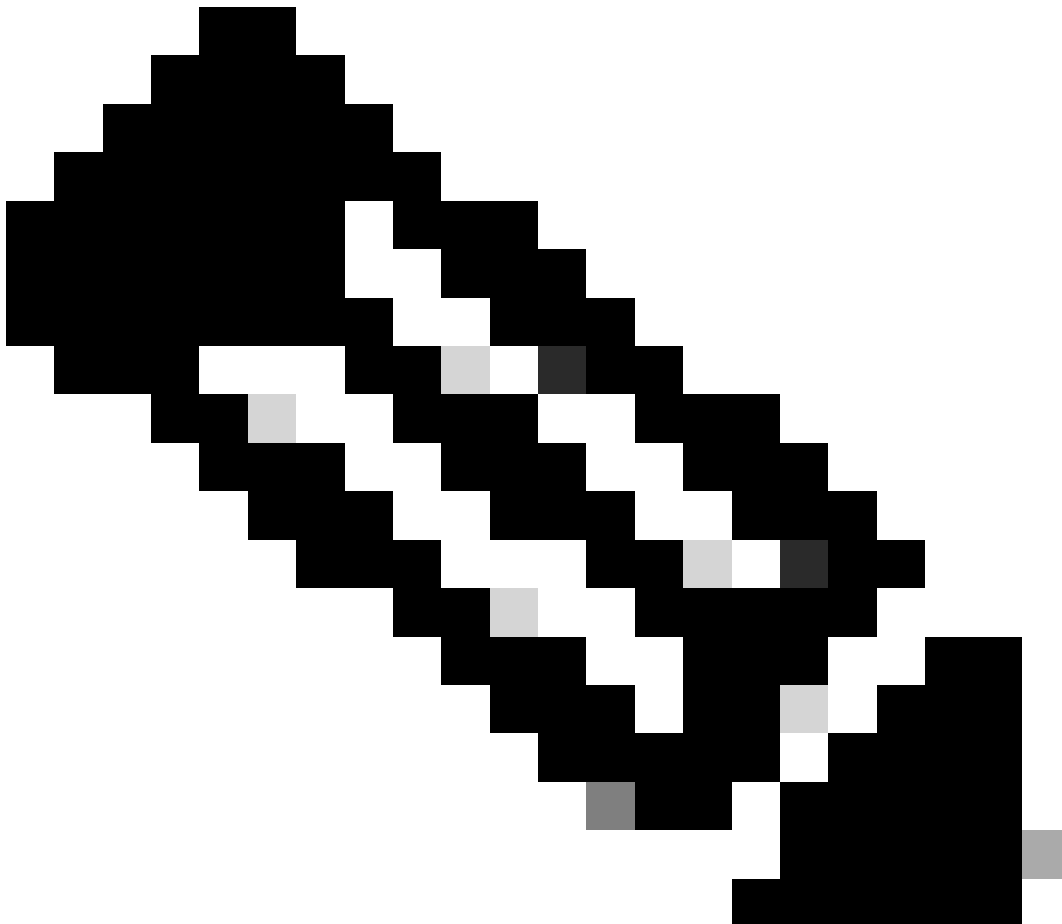
v Algorithms
Cookie: 47a96215afc92003180b60342970a105
kex_algorithms length: 315
kex_algorithms string [truncated]: curve448-sha512,curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,dif
server_host_key_algorithms length: 123
server_host_key_algorithms 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.

```

334 6.093250 10.106.51.72 10.65.54.8 SSHv2 366 Server: Key Exchange Init
> Frame 334: 366 bytes on wire (2928 bits), 366 bytes captured (2928 bits) on interface 0
> Ethernet II, Src: Cimsys_33:44:55 (00:11:22:33:44:55), Dst: Cisco_3c:7a:00 (00:05:9a:3c:7a:00)
> Internet Protocol Version 4, Src: 10.106.51.72, Dst: 10.65.54.8
> Transmission Control Protocol, Src Port: 22, Dst Port: 56127, Seq: 20, Ack: 1285, Len: 312
✓ SSH Protocol
  ✓ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
    Packet Length: 308
    Padding Length: 4
    ✓ Key Exchange
      Message Code: Key Exchange Init (20)
      > Algorithms

```

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.

```

337 6.201114 10.65.54.8 10.106.51.72 SSHv2 326 Client: Diffie-Hellman Group Exchange Init
> Frame 337: 326 bytes on wire (2608 bits), 326 bytes captured (2608 bits) on interface 0
> Ethernet II, Src: Cisco_3c:7a:00 (00:05:9a:3c:7a:00), Dst: Cimsys_33:44:55 (00:11:22:33:44:55)
> Internet Protocol Version 4, Src: 10.65.54.8, Dst: 10.106.51.72
> Transmission Control Protocol, Src Port: 56127, Dst Port: 22, Seq: 1309, Ack: 612, Len: 272
✓ SSH Protocol
  ✓ SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
    Packet Length: 268
    Padding Length: 6
    ✓ Key Exchange
      Message Code: Diffie-Hellman Group Exchange Init (32)
      Multi Precision Integer Length: 256
      DH client e: 1405ab00ff368031363467ad6653967d5a64eac4734e5dc6...
      Padding String: 5c81f2cffc95

```

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)

```

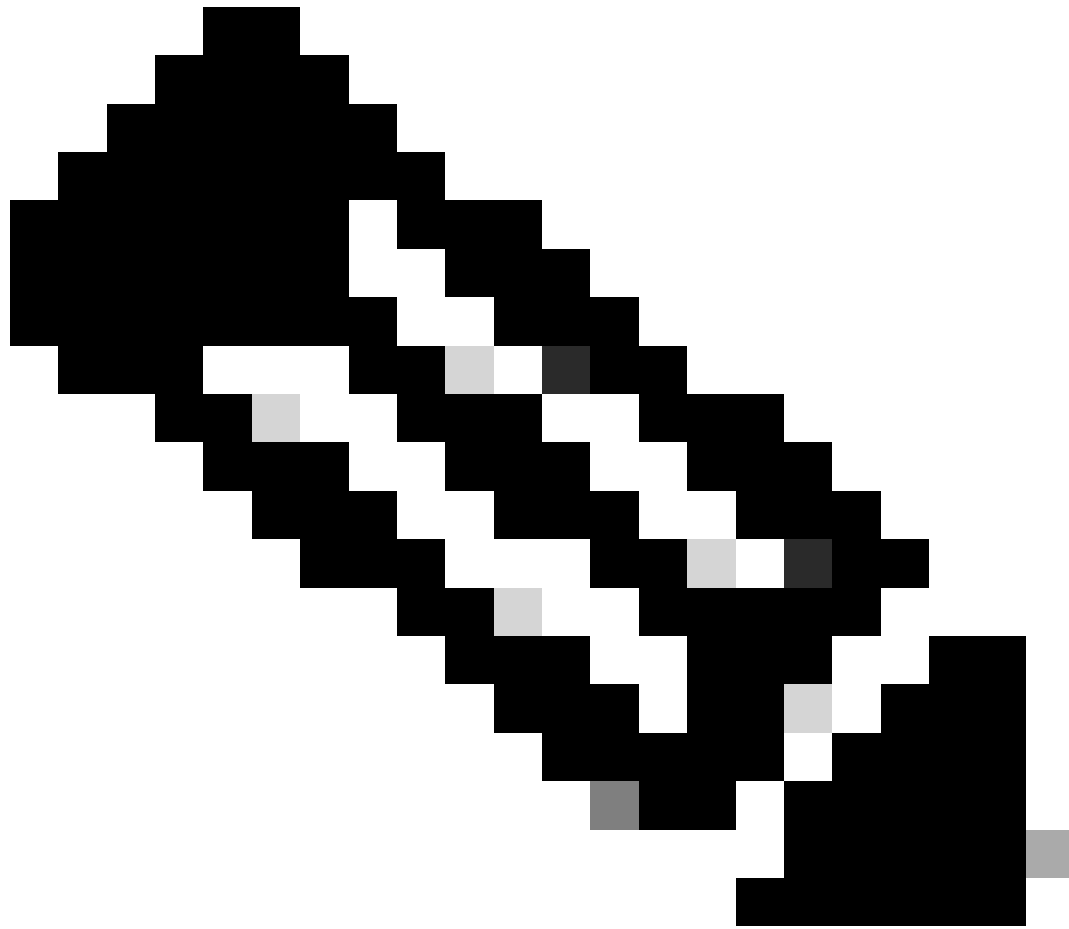
343 6.330017    10.106.51.72    10.65.54.8      SSHv2    350 Server: Diffie-Hellman Group Exchange Reply
Internet Protocol Version 4, Src: 10.106.51.72, Dst: 10.65.54.8
Transmission Control Protocol, Src Port: 22, Dst Port: 56127, Seq: 1148, Ack: 1581, Len: 296
[2 Reassembled TCP Segments (832 bytes): #342(536), #343(296)]
SSH Protocol
  SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
    Packet Length: 828
    Padding Length: 8
    Key Exchange
      Message Code: Diffie-Hellman Group Exchange Reply (33)
      KEX host key (type: ssh-rsa)
        Host key length: 279
        Host key type length: 7
        Host key type: ssh-rsa
        Multi Precision Integer Length: 3
        RSA public exponent (e): 010001
        Multi Precision Integer Length: 257
        RSA modulus (N): 0098c7d23c9ababd730f07b5c2aee1e4e51bac67970aa5af...
        Multi Precision Integer Length: 256
        DH server f: 3a17a0995531f12d629a48ab6f25715bc181ea3deb6c6793...
        KEX H signature length: 271
        KEX H signature: 000000077373682d72736100000100691d2c896761bc7481...
        Padding String: 0000000000000000

```

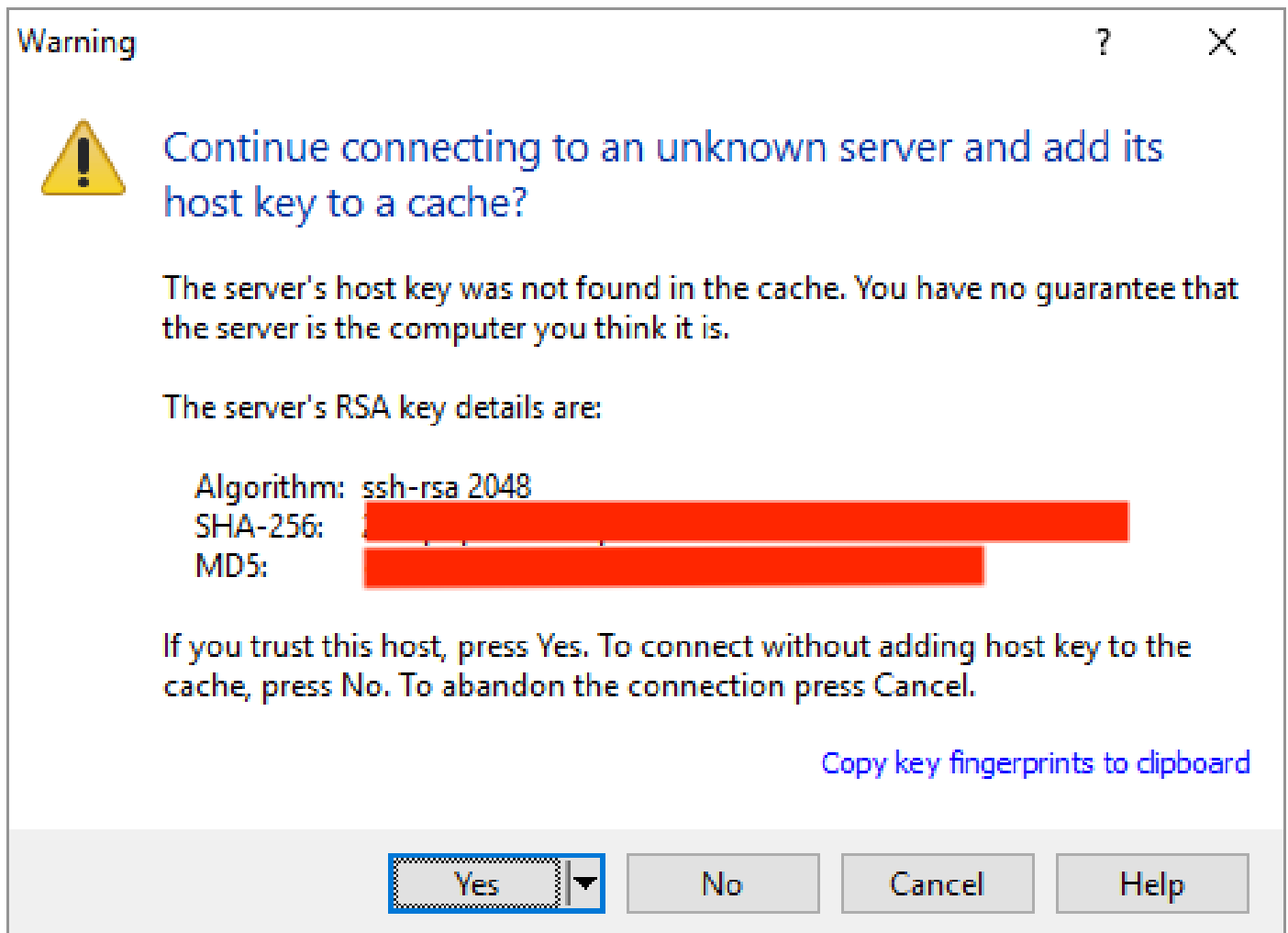
Server Diffie-Hellman Group Exchange Reply

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.

346	6.330368	10.106.51.72	10.65.54.8	SSHv2	70 Server: New Keys
347	6.365552	10.65.54.8	10.106.51.72	SSHv2	70 Client: New Keys

> Frame 346: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0

> Ethernet II, Src: Cimsys_33:44:55 (00:11:22:33:44:55), Dst: Cisco_3c:7a:00 (00:05:9a:3c:7a:00)

> Internet Protocol Version 4, Src: 10.106.51.72, Dst: 10.65.54.8

> Transmission Control Protocol, Src Port: 22, Dst Port: 56127, Seq: 1444, Ack: 1581, Len: 16

> SSH Protocol

- SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
 - Packet Length: 12
 - Padding Length: 10
 - Key Exchange
 - Message Code: New Keys (21)
 - Padding String: 000000000000000000000000

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](#)