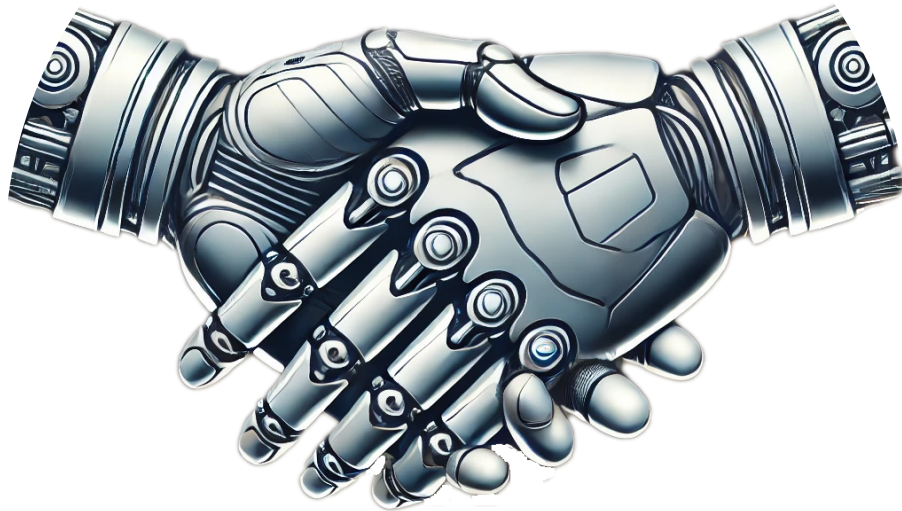


SSH Handshake



- Federico Williamson
- Leonel Castinelli

Seguridad Informática 2024

Índice

I. Introducción	3
I.I. ¿Qué es SSH?	3
I.II. Descripción técnica del problema que soluciona.	3
I.III. Principales aplicaciones.	3
I.IV. ¿Qué es SSH Handshake?	3
II. Intercambio de Versiones.	4
II.I. Cadena de Version ¹	4
II.II. Protocolo binario de paquetes	4
III. Intercambio de algoritmos (KEXInit) ²	5
IV. Intercambio de claves Diffie Hellman ³	6
IV.I. Markus	9
IV.II. Solicitud de servicio ⁴	9
IV.III. Solicitud de Conexion ⁵	10
IV.IV. Ejecucion ⁶	10
V. Apéndice II	11
Bibliografía	13

¹[RFC4253-4.2](#)

²[RFC4253-7.1](#)

³[RFC4253-8](#)

⁴[RFC4252](#)

⁵[RFC4252](#)

⁶[RFC4254](#)

I. Introducción

I.I. ¿Qué es SSH?

SSH (o **Secure SHell**) es el nombre un protocolo y del programa que lo implementa cuya principal función es el **acceso remoto** a un servidor por medio de un **canal seguro** en el que toda la información está cifrada. El puerto TCP asignado de forma predeterminada es el puerto **22**, asignado por la IANA [1].

I.II. Descripción técnica del problema que soluciona.

Ante la comunicación entre dos equipos informáticos, existe la posible amenaza de que un equipo malicioso esté escuchando la comunicación legítima y busque robar información que se esté transmitiendo o incluso intente robar la identidad de uno de los actores en la comunicación para inducir mensajes maliciosos en la misma.

I.III. Principales aplicaciones.

Ya que este protocolo prueba ser de extrema utilidad, se aplica en las siguientes ocasiones:

- **Acceso Remoto Seguro:** Permite controlar equipos informáticos de forma segura a través de internet.
- **Túneles SSH:** Redirección de tráfico en la red a través de un canal seguro, útil para evitar restricciones de red y proteger datos sensibles.
- **Transferencia segura de archivos:** Los protocolos SCP y SFTP utilizan el protocolo SSH para la transferencia de archivos de forma s

I.IV. ¿Qué es SSH Handshake?

Es un proceso por el cual se establece una conexión segura entre un cliente y un servidor. Se produce tras la primera comunicación con el servidor. Consta de **5 pasos**:

- **Paso 1:** Intercambio de Versiones.
- **Paso 2:** Intercambio de Claves (KEX).
- **Paso 3:** Inicialización de Elliptic Curve Diffie-Hellman (ECDH).
- **Paso 4:** Respuesta de ECDH.
- **Paso 5:** Nuevas Claves (NewKeys).

II. Intercambio de Versiones.

II.I. Cadena de Version⁷

La cadena de versión de un participante en la comunicación se encuentra en el siguiente formato:

`<SSH-protocolversion>-<softwareversion> <comments>CRLF`

Con esta cadena en este formato ambos participantes podrán intercambiar la versión del protocolo que están utilizando y la versión de software que están utilizando. Y siempre deben terminar con CRLF⁸.^[2]

Ambas partes, **cliente** y **servidor** deben enviar sus cadenas de versión. Las cuales llamaremos V_C a la cadena de version del **cliente** y V_S a la del **servidor**.

```
INFO      SSHClient: Connected to server
DEBUG     SSHClient: My SSH version: SSH-2.0-LeoFedeSsh0.1 Un trabajo para un tp
DEBUG     SSHSocketWrapper: c >> s [46]: 5353482d322e302d4c656f4665646553 7368302e31205556e2074726162616a6f 207061726120756e2074700d0a
DEBUG     SSHSocketWrapper: s >> c [21]: 5353482d322e302d4f70656e5353485f 392e300d0a
DEBUG     SSHClient: Remote SSH version: SSH-2.0-OpenSSH_9.0
```

Acá podemos observar como enviamos nuestra versión y recibimos la versión del servidor.

II.II. Protocolo binario de paquetes

Cada paquete se encuentra en el siguiente

```
uint32    packet_length
byte      padding_length
byte[n1]  payload; n1 = packet_length - padding_length - 1
byte[n2]  random padding; n2 = padding_length
byte[m]   mac (Message Authentication Code - MAC); m = mac_length
```

Donde cada parte representa:

- **packet_length**: Es la longitud del paquete en bytes, sin contar “mac” o el mismo campo “packet_length”
- **padding_length**: longitud del “random padding” en bytes.
- **payload**: La parte útil del paquete.
- **random_padding**: padding de longitud arbitraria, para que la longitud total (packet_length + padding_length + payload + random_padding) sea un múltiplo de 8, el padding tiene que ser cómo mínimo de 4 y máximo de 255. Además sirve para introducir ruido al mensaje, confundiendo a receptores ilegítimos de la comunicación.
- **MAC**: Código de Autorización de Mensajes, si se negoció la autenticación de mensajes, este campo contiene los bytes de MAC. Inicialmente el algoritmo de MAC es none.

II.II.I. MAC

Una funcionalidad de proteccion, de extrema utilidad en lo que concierne a la seguridad de la comunicación, que ofrece SSH en su modelo de paquetes es MAC: Message Authorization Code (Código de Autorización de Mensajes).

Este codigo es un algoritmo de hash del contenido del mensaje no encriptado, con un número de secuencia del paquete.

Esto es extremadamente util en evitar replay attacks.

⁷RFC4253-4.2

⁸CRLF es un \r seguido de \n

III. Intercambio de algoritmos (KEXInit)⁹

```
Local key:
SSHKEXInitPacket(
    cookie=b'$\x1a\x7fM\x86E!\xafH\x14X\xfd<\xf8\x90\x93',
    kex_algorithms=[b'diffie-hellman-group14-sha256'],
    server_host_key_algorithms=[b'rsa-sha2-256'],
    encryption_algorithms_client_to_server=[b'aes128-ctr'],
    encryption_algorithms_server_to_client=[b'aes128-ctr'],
    mac_algorithms_client_to_server=[b'hmac-sha1'],
    mac_algorithms_server_to_client=[b'hmac-sha1'],
    compression_algorithms_client_to_server=[b'none'],
    compression_algorithms_server_to_client=[b'none'],
    languages_client_to_server=[b''],
    languages_server_to_client=[b''],
    first_kex_packet_follows=False,
    reserved=0
)
```

Nosotros mandamos este paquete de tipo KEX init al servidor.

Aca manifestamos los algoritmos que soportamos para cada categoria, en orden descendente de preferencia. Para forzar el uso de ciertos algoritmos y simplificar la implementación, enviamos listas de un único elemento.

```
Remote key:
SSHKEXInitPacket(
    cookie=b'\x01\xf0\xff\xfa\x01\x06\xdd\x88\x8b6Q\x1e6\x12\x81/',
    kex_algorithms=[
        b'sntrup761x25519-sha512@openssh.com',
        b'curve25519-sha256',
        b'curve25519-sha256@libssh.org',
        b'ecdh-sha2-nistp256',
        b'ecdh-sha2-nistp384',
        b'ecdh-sha2-nistp521',
        b'diffie-hellman-group-exchange-sha256',
        b'diffie-hellman-group16-sha512',
        b'diffie-hellman-group18-sha512',
        b'diffie-hellman-group14-sha256'
    ],
    server_host_key_algorithms=[b'rsa-sha2-512', b'rsa-sha2-256', b'ecdsa-sha2-nistp256', b'ssh-ed25519'],
    encryption_algorithms_client_to_server=[b'chacha20-poly1305@openssh.com', b'aes128-ctr', b'aes192-ctr', b'aes256-ctr', b'aes128-gcm@openssh.com', b'aes256-gcm@openssh.com'],
    encryption_algorithms_server_to_client=[b'chacha20-poly1305@openssh.com', b'aes128-ctr', b'aes192-ctr', b'aes256-ctr', b'aes128-gcm@openssh.com', b'aes256-gcm@openssh.com'],
    mac_algorithms_client_to_server=[
        b'umac-64-etm@openssh.com',
        b'umac-128-etm@openssh.com',
        b'hmac-sha2-256-etm@openssh.com',
        b'hmac-sha2-512-etm@openssh.com',
        b'hmac-sha1-etm@openssh.com',
        b'umac-64@openssh.com',
        b'umac-128@openssh.com',
        b'hmac-sha2-256',
        b'hmac-sha2-512',
        b'hmac-sha1'
    ],
    mac_algorithms_server_to_client=[
        b'umac-64-etm@openssh.com',
        b'umac-128-etm@openssh.com',
        b'hmac-sha2-256-etm@openssh.com',
        b'hmac-sha2-512-etm@openssh.com',
        b'hmac-sha1-etm@openssh.com',
        b'umac-64@openssh.com',
        b'umac-128@openssh.com',
        b'hmac-sha2-256',
        b'hmac-sha2-512',
        b'hmac-sha1'
    ],
    compression_algorithms_client_to_server=[b'none', b'zlib@openssh.com'],
    compression_algorithms_server_to_client=[b'none', b'zlib@openssh.com'],
    languages_client_to_server=[b''],
    languages_server_to_client=[b''],
    first_kex_packet_follows=False,
    reserved=0
)
```

La respuesta del servidor, mostrando todos los algoritmos que soporta para cada categoria.

El protocolo SSH define que se usa el primero en comun, llenando en orden de izquierda a derecha.

Es posible que en cada direccion se usen algoritmos distintos

En este momento, ambos cliente y servidor saben que algoritmo van a usar para cada caso (lo pueden inferir por separado)

A los **payload** de **cliente** y **servidor** los llamaremos I_C y I_S respectivamente

⁹RFC4253-7.1

IV. Intercambio de claves Diffie Hellman¹⁰

Nosotros, elegimos usar el algoritmo de diffie-hellman-group14-sha256 como algoritmo de intercambio.

Esto significa que:

- Vamos a hacer un intercambio de tipo DH. (que usará Curvas Elípticas Diffie-Hellman, ECDH)
- Vamos a usar primos del grupo 14.
- Vamos a usar sha256 como algoritmo de hash.

El objetivo del KEX DH es que ambos participantes puedan mutuamente acordar en una clave igual, sin que esta sea derivable por un tercero que esta observando la comunicación.

Como funciona esto?

1. Se acuerdan p y g . P es un primo publico muy grande, y g su generador.

Como estamos usando group14, usamos el primos del grupo 14, que esta definido en [RFC3526-3](#)

C (cliente) genera un numero aleatorio x entre 1 y q^{11} y computa el valor:

$$e = g^x \bmod p$$

El cual es conocido como **clave pública del cliente** y a x lo llamaremos **clave privada del cliente**.

```
SSHClient: Generating keys...
SSHClient: P:
32317080071310073083389129264238282488179412414023911284208975148074176663435422261968941736356934711796173798970410175460587320919500885375888618562215321217541251498177452027
0235796782362488842461894775876411059286468994172245426622522193230548919037680524235519125679715870117001058053877651038861847280257976054903569732561526167081339361799541336
476559160368317896729073178384589680639671900977202194168642725871031411336429319536193471636533209717077448227988588553692086452966360772502689555059283627511211740969729980684
1055435958486658291642136218231078990999448652468262416972035911852507045361090559
00000000: ff ff ff ff ff ff ff ff c9 0f da a2 21 68 c2 34 .....h.4
00000010: c4 c6 e2 0b 80 dc 1c d1 29 02 4e 08 8a 67 cc 74 .....b....)M.g.t
00000020: 02 0b be a6 3b 13 9b 22 51 4a 08 79 8e 34 04 dd .....Q.Y.4..
00000030: ef 95 19 b3 cd 3a 43 1b 30 2b 0a 6d f2 5f 14 37 .....C.+.m...7
00000040: 4f e1 35 6d 6d 51 c2 45 e4 85 b5 76 62 5e 7e c6 0.5mmQ.E...vb^w.
00000050: f4 4c 42 e9 a6 37 ed 6b 0b ff 5c b6 f4 06 b7 ed ...LB-7K.v....
00000060: ee 38 6b fb 5a 89 9f a5 ae 9f 24 11 7c 4b 1f e6 ...8k.Z....$.K..
00000070: 49 28 66 51 ec e4 5b 3d c2 00 7c b8 a1 63 bf 05 ...(fQ..[=..]...c..
00000080: 98 da 48 36 1c 55 d3 9a 69 16 3f a8 fd 24 cf 5f ...HG.U..i.f.$...
00000090: 83 65 5d 23 dc a3 ad 96 1c 62 f3 56 20 85 52 bb ...0J.....b.V.R.
000000a0: 9e d5 29 07 70 96 96 6d 67 0c 35 4e 4a bc 98 04 ...).p..mg.5Nj..
000000b0: f1 74 6c 08 ca 18 21 7c 32 90 5e 46 2e 36 ce 3b ...t1...l|2.F.G.;
000000c0: e3 9e 77 2c 18 0e 86 05 9b 27 83 a2 ec 07 a2 8f ...w.....
000000d0: b5 c5 3d f0 6f 4c 52 c9 de 2b cb f6 95 88 17 18 ...j..oLR...+..X..
000000e0: 39 95 49 7c ea 95 6a e5 15 d2 26 18 98 fa 05 10 9.i|.j...&.....
000000f0: 15 72 8e 5a 8a ac aa 68 ff ff ff ff ff ff ff ff ...r.Z...h.....

SSHClient: G: 2
00000000: 02

SSHClient: X: 2516780003010373300104893832166956649264932701074034475852147363747570402513
00000010: 37 a4 78 52 b7 8c 33 d6 8f 0b 2b 8b 39 16 6c 1e 7.xR...+..9..
00000020: b9 8e c5 66 0d 5e fa 12 d6 b2 b1 b7 c0 64 44 d1 ...f.^......dD.

SSHClient: E:
2878086243511573795961740467762721548135800385689353706991335612647291718022677613608949181947494963984828197098294154143611266621962642345811199308820420598060377240300848054951
909685646849256372756208102760636425451635734032565264765233437786116721760654972492111825900898657037092212011872281514377272287484317076036782883562839024500055445191390647
7850934196562949122418273928194471511337493357046207620649821273871658359503102174902232556217442019522561540980864058049267259585927243478947068975716377257379021101730532187751
70462721811693308129176920748640527104952932254556409888686467904465049151715896572
00000000: e3 fd 04 49 7f 18 66 9d 21 93 4a 86 d7 5d da e3 ...T..f.i.j..j..
00000010: d5 36 bd 45 24 43 ac 05 a9 39 1e 64 2c 01 16 a1 ..6.E$CL..9.d...
00000020: 06 23 5c ab af bb c2 a9 3d 7c 46 16 a4 54 ca d6 ...#\.....=F..T...
00000030: 2c 93 eb bc f2 b5 7a cf 62 e2 d8 08 32 3c d8 de .....z.bcc..2C..
00000040: 80 c6 02 6a 46 f5 00 8b c9 47 4c a7 41 55 90 19 ...jF.....GLAu..
00000050: 2c 0e 81 38 24 5e 99 9e 76 20 41 3b cf 07 95 29 ...8$^..v.A;....
00000060: 45 40 2f ad ee 46 c9 fe 12 d9 ca 2c 82 c1 b5 62 E0/.F.....b
00000070: 90 29 cb cf 4e 4b ed 63 a5 93 fc ce 66 50 ee 33 ...).INK.c....fp.3
00000080: d1 b0 d1 c8 d9 f0 4c fa 14 8e cb 48 47 65 50 00 .....jF.....HdP.
00000090: 1d b4 33 aa 59 d5 58 df 21 72 42 27 4a bb 4f 61 ...3.Y.X.lrbJ.Oa
000000a0: 86 37 e4 39 1c ed 75 27 2a 77 4d 1b 13 b3 14 02 ...7.9..u'hm.....
000000b0: 88 10 9b f9 2c 54 40 d5 63 39 d9 f5 71 c1 b3 0d ....jT0.c9...q...
000000c0: 8d a7 7e c2 29 35 b7 73 6c 65 f4 9d f5 67 d9 5d ...e).5.sle....e.1
000000d0: 94 f0 49 4b 84 ae 6a ec 67 a4 d0 ca 1e ae 1d 02 ...IK.....g.....
000000e0: 08 3f 33 63 6a a1 dd 9e f2 f7 82 76 9b 8e 96 e4 ...73cj.....v....
000000f0: 46 06 1b c6 9b 74 93 66 04 f2 a7 fb f8 ff 6c fc F....t.f.....l.
```

S (servidor) genera su propio numero aleatorio y entre 0 y q y computa:

$$f = g^y \bmod p$$

Acá se puede observar la **clave pública del servidor** f y la **clave privada del servidor** y .

¹⁰[RFC4253-8](#)

¹¹ q hace referencia al orden del subgrupo, que no se necesita clacular explicitamente, es aproximadamente $\lfloor \frac{p}{2} \rfloor$

```

SSHPacket: Creating packet with length 276, padding length 13 (13), payload length 262 - 8
SSHSocketWrapper: c >> s (280): 0000011401000001010003fd04a9ff 18669021934a86175ddae3d536bd452a 434c05e9391e642c9116a106235cabaf bbc2a93d7c4616a454cad62c93ebbcf2
b57acf62e4d808323cd8de80c6026a46 f5b08bc9474c7415590192cde81382a 5e999ef620413bcf97952945402fade 46c9fe12d9ca2c82c1b5629029cbaf4e 4bed63a593fcee650ee33d1b0d1c8d9
f04cfa148ecb48476550001db433aa59 d558df217242274abb4f618637e4391c ed75272a774d1b13b3140288109bf92c 5440d56339d9f571c1b30d8da87ec229 35b7736c65fd9df567d59d4f0494b84
e08aec67e4d0ca1ee1d02083f33636a a1dd9ef2f782769b8e96e446061bc69b 74936604f2a7fbf81f6cf808dec37641 86bcb2af9048d9e
SSHClient: Sent SSH_MSG_KEXDH_INIT
SSHClient: waiting for server response
SSHSocketWrapper: s >> c [4]: 00000444
SSHSocketWrapper: s >> c [1]: 0a
SSHSocketWrapper: s >> c [1081]: 1f0000019700000077373682d727361 0000000301000100000181009c650762 b31c7096e3786c5c967b30245452cf22 22d6f363dd0f56ed6cb8686579a0deaf
82eb0b4144eb05fcd38782d08d9b6da1b b152bdfa4c4e813f94d68cb6f446f85c 81929cdeb32589e78e029473811217093 b191cb9e4b586b986834fa2d422a25f0 31d4a876a7a17a2eae7ffb9192cd314a
ae78d9a6db8a2d8cf6c74fec11b420 1f5c4ef7bad2463d568c1ddbafd1bb96 5c177227734fa3d1e6546e5d0f1b646 83a899fe7acb2f6d5f5648c53bbca00 60813d27e9132633ad995bcb5e48d106b
d1211bf0e70801c4f6b9397aa48c016 66c601c9600740bf231881cdc410433c 0850339214716ab789ac329b6d2ed21 26d94593d62909be4413ff5bad8fb651 fa3eb4ade30448f9db42150a243ea304
e3b9adaa88599b1441c9bf813c614dc6 3889482af8598f0c27143795316bdb4d a7f93d3f4cac56e632e8d747b73c22c2 226ae9a15fd7b3c30ff7402534650951 8e979f8aed03ae5fbhbd5f32ea7c76c9b
b3c6963c8aa5d76289370c56831dcef3 6d3d67011f7b2c26616e72870000101 0080c3cf6bdde709a3866a1783bbeb39 a365d268b0594a8d7505570c014f1a5e 641a43eb09face97fc80af46dcd94e35
b42e2b4ec3ba9a516e0d1ca5f7a81d930 9e170f4d672931fb8dc9ce7e02b4f5e9 2a87f01d4f3e8bed99e1632d3380238c 4c9e831d25719dfb56e19edd35d087bf 60dd352b59a91c0cd6554d8d1047c50d
b307d749611e5e4535974a425e8b1b07 605102aa0864ab266f432302d78289f9 e9db700718594d0dd421208abd2c446f 0db8f114ed6dcfab9dfedaf197b8773f 34edbe74e5544dc51a27b3fbee752c554
6034bf74e26713357f022949379fcb0b4 6f4ebd6cbe2d7c66bf944c9ee1e1c8 bc84a1b24b0e25af82b3d117f681f8d12 fc0c00019400000007273612d736861 32d23235360000018063aa54a2ubee27a
fa00b1c39d2cab41b4d05f3211be9f5d 9a6ae5e4dd875be17bef55850e0502c8 1ebcc512a3694c69c72d2eed192b2f33 62c297c6cef0dcdbde60ealc804df5987 d5888fcd3b3132d3aaeabcb42d5bbf190
0f270ecdad9d0aa31722c1a58f563a71 430e1e7f973d87f16075fc40d5d37aec f9ef08773080db3db8603243732d1a dddde8c994208c0915334f9ea106ac82 347c0330e6e1efe596e68215735ce6c1
a864997d579032504bc9feal75c64081 0040d12e24f7cf25faac8c2dd481f4f0 3643bebdfec2da196d92906e8858dd4c 95632c5de4abfed4300813d52050698d eb91d86878035daced59dabc4bd053fa
da92d2e2d325c0f01e465f0b82879ce5 c5c2bcedc22e2960f99e749c060c6721 eab812e1d2fa7574af8a946819bda512 98a272acf953cated34135baae908b 51f0c18383fb017c663af7e9ba546a47
SSHSocketWrapper: s >> c [10]: 00000000000000000000
SSHSocketWrapper: Received packet[1]: 31 (SSH2_MSG_KEXDH_REPLY)
SSHClient: Server f value:
00000000: 80 c3 cf 6b dd e7 09 a3 86 6a 17 83 bb eb 39 a3 ...k.....j....9.
00000010: 65 d2 68 b0 59 4a 8d 75 05 57 0c 01 4f 1a 5e 64 ...e.h.YJ.u.W..O.d
00000020: 1a 43 eb 09 fa ce 97 fc 80 af 46 dc d9 4e 35 b4 ...c.....F..NS.
00000030: 2e 2b 4e c3 b9 a5 16 e0 d1 ca 5f 7a 81 d9 30 9e ...+N.....z..0.
00000040: 17 0f 4d 67 29 31 fb 8d c9 ce 7e 02 b4 f5 e9 2a ...Mg)1.....~..*
00000050: 87 f0 1d 4f 3e 8b ed 99 e1 63 2d 33 80 23 8c 4c ...O>.....c-3.#.L
00000060: 9e 83 1d 25 71 9d fb 56 e1 9e dd 35 d0 87 bf 60 .....Sq..V...5....
00000070: dd 35 2b 59 a9 1c 0e d6 55 4d 8d 10 47 c5 04 b3 ...s.YV.....UM..G...
00000080: 07 7d 49 61 1e 5e 45 35 97 4e 42 5e 8b 1b 07 60 ...}Ia~ES.NB~...
00000090: 51 02 aa 0d 64 ab 26 6f 43 23 02 d7 82 89 f9 e9 ...Q...d.&cW~.....
000000a0: db 70 07 18 59 4d 0d d4 21 20 0a bd 2c 44 6f 0d ...p..YM...!...Do.
000000b0: b8 f1 14 ed 6d cf ab 9d fe da f1 97 b8 77 3f 34 .....m.....w24
000000c0: ed be 7a c5 94 4d c5 1a 27 b3 fb e7 52 c5 54 60 ...t.TM.....R.T'
000000d0: 24 bf a4 26 71 33 5f 72 92 94 93 79 fc b0 b4 6f ...A..RqW~...v.y.o
000000e0: 4e bd 6c cb e2 d7 c6 e6 bf 54 4c 9e e1 e1 c8 bc ...N.L.....TL.....
000000f0: 84 a1 b2 4e e2 50 af 82 b3 d1 97 e8 1f 8d 12 fc ...K.P.....

```

S al recibir e, computa el **secreto compartido** K de la siguiente forma:

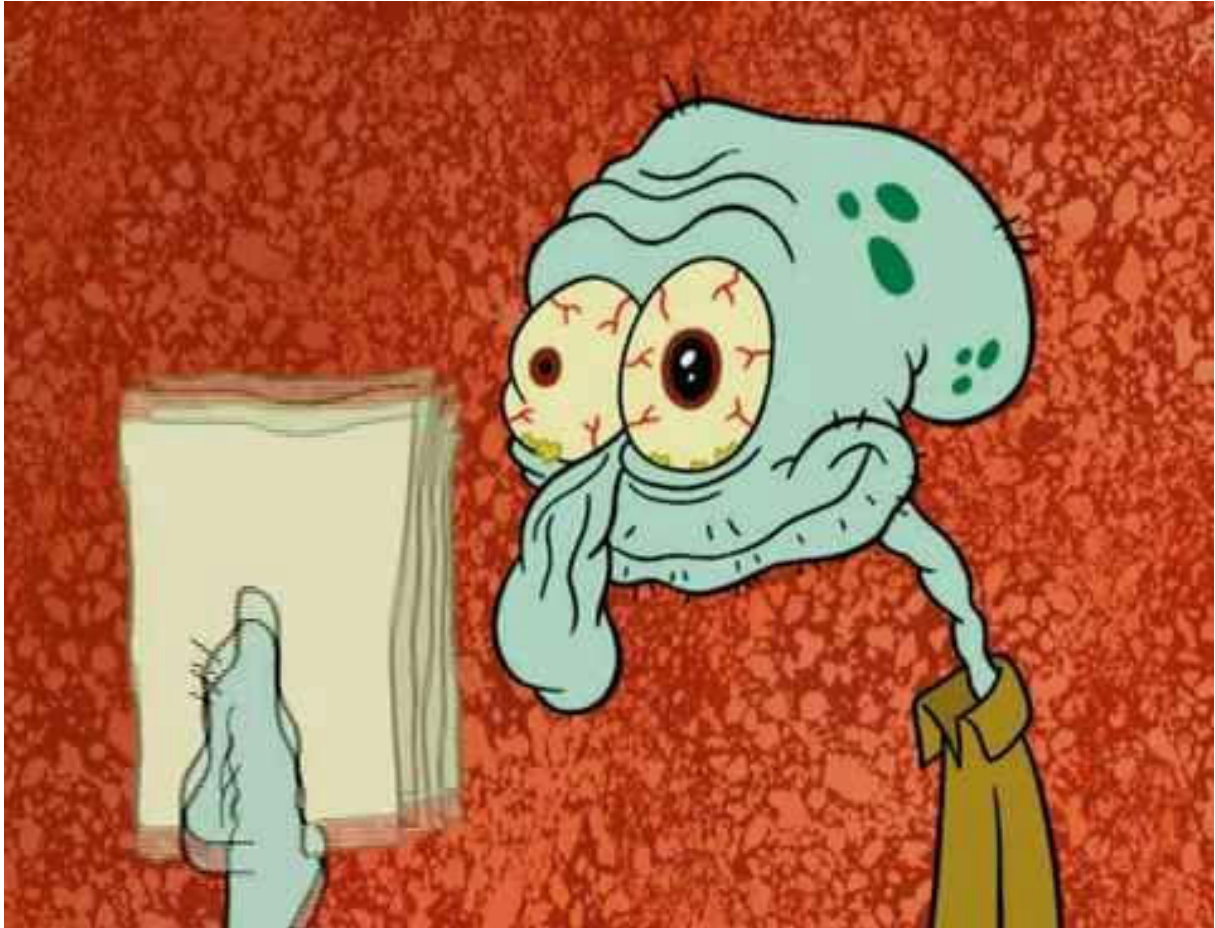
$$K = e^y \bmod p$$

Una vez que ves. recibe x del C, ya esta listo y manda SSH_NEW_KEYS. Cuando C recibe f de S, puede emitir SSH_NEW_KEYS.

```

SSHSocketWrapper: Received packet[2]: 21 (SSH2_MSG_NEWKEYS)
SSHClient: New keys: SSHNewKeysPacket()
SSHPacket: Creating packet with length 20, padding length 18 (18), payload length 1 - 8
SSHSocketWrapper: c >> s [24]: 000000141215f6a12bd2eb47f62868 04a2f01dd192638
SSHClient: Habemus New Keys!
SSHClient: Derived keys:
1048001053212530762997067268572211043268041418717053192897712724197860274709508478181893107373778918309912406272946974005261845297056974166883760035153879751178777157620995155168
13344924165988736957749026294048861725948663001157941527019759587529341370612811256603872214070103509725383067506791489033737961502653153055461288005312474321005020581477475
9506421912112351866848759636405991249602768742392276468200126270479929878250836395433632323417802030502349150020084887265524957040628702310676228845406732344036946540836450102
179613786418258907986867712724260681492611280450145579596299888131568042124739490
00000000: c3 04 80 53 d4 b3 53 cf 7c ed 52 7e 62 61 f2 f2 S...S...S..l..R..bc...
00000010: af 2f 13 b9 74 92 ac 1c b7 ca af 2b 92 90 85 8a .../.t.....t....
00000020: e0 62 83 e4 8f 3c 27 0c fa 2c 6c b0 73 81 f3 cc ...b...<...l.s...
00000030: 4a 3d c2 97 bb 41 5e 12 ae 0b 42 56 3e f7 40 63 ...e...A~...Bb6.g...
00000040: c9 f5 9e 0d 28 35 84 97 00 93 79 08 a7 40 af ac ...S.....y..HO.
00000050: 31 46 6e 95 f0 0a 22 6e cf 47 2d 57 2d 5c a7 46 1Fn...n.G-W-V..F
00000060: bf 07 d0 00 50 56 11 95 da 55 1c 80 b2 70 8f 41 ...PV...U...J..A
00000070: 8c 5a 93 87 3f 2e f4 75 c1 8c aa bb a0 43 38 76 ...~?..u.....GBV
00000080: fd 67 5a c2 f8 10 3c 0c 80 1f 52 9f 2f a7 9e 3c ...gZ...<...R../<
00000090: e1 8c 05 8a 0b 3a 0f a1 f7 9a 0d 05 1c a4 d8 d8 ...t.....
000000a0: 6f 93 a5 b7 c5 09 2d 3d 4d 48 06 9e 80 7e ba o...&..]~Wk...
000000b0: dd 24 10 80 56 3f 86 55 73 ed 24 b5 f6 7a d0 da ...$.V?..Us..$.Z...
000000c0: 98 f2 98 81 f6 07 8c 0c 57 fe ff c6 fd 64 95 e3 ...W.....
000000d0: df 47 31 70 aa 1d 2b 36 f3 55 0a 1c 07 48 51 44 ...Gix...46.U...HQ0
000000e0: c8 b5 61 b3 90 a5 00 26 c1 ca 76 ee 1b bf 09 fc ...a.....&..v....
000000f0: 74 71 f4 81 19 e4 e5 62 48 5d 29 ae 6e 91 ab a2 tq.....btj}).n...

```

En este momento ambos tienen suficiente información para terminar de derivar la clave compartida K , pero únicamente K es insuficiente para la comunicación

y también computa el hash H :

$$H = \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel e \parallel f \parallel K)$$

Donde:

- hash es sha256, por el algoritmo de intercambio que elegimos,
- V_C Es la versión de ssh cliente (sin CRLF)
- V_S Versión de ssh del servidor (sin CRLF)
- I_C El Payload del KEX_INIT enviado por el cliente
- I_S El Payload del KEX_INIT enviado por el servidor
- K_S La Host-Key del servidor¹²
- e La clave pública del cliente
- f La clave privada del servidor
- K el secreto compartido entre las partes.

Un dato interesante de esto, es que al incluir el payload de I_C e I_S , que como anteriormente cubrimos en Sección III, contienen un cookie que contiene un valor random. Esto hace que sea completamente imposible para un atacante determinar H en sesiones repetidas.

En base a H , K y la session id¹³ se calculan los siguientes valores:

¹²Este es un valor adicional que envía el servidor cuando envía f , con el propósito de autenticarse.

¹³La session id es el primer H que se acuerda entre C y S, y no cambia cuando hay un reset de keys.

- o Initial IV client to server: HASH(K || H || "A" || session_id)
(Here K is encoded as mpint and "A" as byte and session_id as raw data. "A" means the single character A, ASCII 65).
- o Initial IV server to client: HASH(K || H || "B" || session_id)
- o Encryption key client to server: HASH(K || H || "C" || session_id)
- o Encryption key server to client: HASH(K || H || "D" || session_id)
- o Integrity key client to server: HASH(K || H || "E" || session_id)
- o Integrity key server to client: HASH(K || H || "F" || session_id)

IV.I. Markus

Para probar el cliente, con el servidor de openssh, es posible ejecutarlo en modo debug. Una de las funcionalidades del modo debug, es una vez que se inicia la comunicacion encriptada, se envia un paquete de tipo 02 (SSH_IGNORE) con texto: markus. Este mensaje conocido es extremadamente util para debuggear problemas.

markus en este caso hace referencia a Markus Friedl, un contribuidor original de SSH.

```

DEBUG      SSHSocketWrapper: s >> c [5]: 6cfd30b2ea
DEBUG      SSHPacket: length=28 pad=16
DEBUG      SSHSocketWrapper: s >> c [27]: bcd48930491313f1eb620d2bbc2927ed 448207c49e8170a51b09f2
DEBUG      SSHSocketWrapper: s >> c [20]: 140c6f551723c2d21cc27d215b72857d 4cc113f0
DEBUG      SSHPacket: [RECV] Encrypted packet:
00000000: 6c fd 30 b2 ea bc d4 89 30 49 13 13 f1 eb 62 0d  1.0.....0I....b.
00000010: 2b bc 29 27 ed 44 82 07 c4 9e 81 70 a5 1b 09 f2  +.)'.D.....p...
00000020: 14 0c 6f 55 17 23 c2 d2 1c c2 7d 21 5b 72 85 7d  ..oU.#.....![r.]
00000030: 4c c1 13 f0                                     L...

DEBUG      SSHPacket: [RECV] Decrypted packet:
00000000: 00 00 00 1c 10 02 00 00 00 06 6d 61 72 6b 75 73  .....markus
00000010: 09 fe 9b d4 72 b0 fd a2 0b 2c 3a 17 fa c5 21 96  ....r.....!..

INFO       SSHClient: Validate MAC #3 on
00000000: 00 00 00 1c 10 02 00 00 00 06 6d 61 72 6b 75 73  .....markus
00000010: 09 fe 9b d4 72 b0 fd a2 0b 2c 3a 17 fa c5 21 96  ....r.....!..

INFO       SSHClient: mac_s2c is
00000000: f5 98 ab 55 d1 5d 04 83 f3 2d 03 c9 3d 9f 5f 0b  ...U.].....=..
00000010: 33 ec c8 0f 17 41 0d 93 69 50 a0 cc e1 b8 ba 87  3....A..iP.....

INFO       SSHClient: MAC VALIDATED with offset 3
DEBUG      SSHSocketWrapper: Received packet[3]: 2 (SSH2_MSG_IGNORE)
SSHPacket(28, 16, b'\x02\x00\x00\x00\x06markus', b'\t\xfe\x9b\xd4\xb0\xfd\xa2\x0b,\x17\xfa\xc5!\x96', 2 (SSH2_MSG_IGNORE))
DEBUG      SSHPacket: Creating packet with length 44, padding length 32 (16), payload length 11 - 16
DEBUG      SSHPacket: [SEND] Unencrypted packet:
00000000: 00 00 00 2c 20 02 00 00 00 06 6d 61 72 6b 75 73  ...., .....markus
00000010: 6a 5e f2 1d 35 d7 f7 87 e5 96 f2 95 8e 5a eb f7  j^..5.....Z..
00000020: 80 eb e8 44 7f c6 43 69 4f db 15 0d 59 5e c2 ea  ...D..CiO...Y^..

DEBUG      SSHPacket: [SEND] Encrypted packet:
00000000: 91 72 58 18 04 5b b7 a0 70 f7 82 99 fd 5d 9c 88  .rX..[.p....]..
00000010: 6e 55 ab ed 54 f9 7d 93 4a dc b0 f8 7c 5d 64 e0  nU..T.}.J....[d.
00000020: dd 74 55 d1 12 0f a3 48 7d d3 5e 1e ef 75 d6 f2  .tU....H}.^..u..
00000030: 1f 7f 15 db 1a ff 5f 85 73 fe c7 a7 14 f7 92 a6  ..._..s.....
00000040: 2f ff 61 91                                     /.a..

```

Aca se puede observar como tambien enviamos un paquete ignorable con markus al servidor para verificar que no produce errores.

IV.II. Solicitud de servicio¹⁴

Antes de proceder, requerimos que el servidor nos permita autenticar usuarios.

¹⁴[RFC4252](#)

```

SSHSocketWrapper: c >> s [68]: 91725818045bb7a070f78299fd5d9c88 6e55abed54f9d934adcb0f87c5d64e0 dd7455d1120fa3487dd35e1eef75d6f2 1f7f15db1aff5f8573fec7a714f792a6
2fff6191
SSHPacket: Creating packet with length 44, padding length 26 (18), payload length 17 - 16
SSHPacket: [SEND] Unencrypted packet:
00000000: 00 00 00 2c 1a 05 00 00 00 0c 73 73 68 2d 75 73 .....ssh-us
00000010: 65 72 61 75 74 68 0d 9b 19 67 bb 25 97 7f f2 f2  erauth...g.%.../
00000020: e3 d5 d8 84 7b 6a 94 35 ce 48 ed e3 cd 2a a0 fc ...{j.S.H...*.

SSHPacket: [SEND] Encrypted packet:
00000000: ec f2 ff 8e ed 23 1d f5 a6 48 4d 56 03 8d 40 e5 .....#.HMV.@.
00000010: 69 b2 a9 24 fc 22 42 ff c2 ef dc bc 64 40 6e 0a i..$. "B.....d@n.
00000020: a9 c3 da d3 0b 5d 2c 5c 66 2e 16 aa d7 2d aa bf .....],f.....
00000030: e1 11 a7 f0 69 60 c4 90 49 d4 8a dd cb e4 5b a8 ....i..I.....[.
00000040: 4f 5e ee d3 ..... 0^..

SSHSocketWrapper: c >> s [68]: ecf2ff8eed231df5a6484d56038d40e5 69b2a924fc2242ffc2efdcbc64406e0a a9c3dad30b5d2c5c662e16aad72daabf e111a7f06960c49049d48addcbe45ba8
4f5eed3
SSHSocketWrapper: s >> c [5]: 591f7a8833
SSHPacket: length=28 pad=10
SSHSocketWrapper: s >> c [27]: 10b86449a5a5c141e8aa5a7b1776abd 94e14ba3ed1b6b6d6980e
SSHSocketWrapper: s >> c [20]: a28fd72a5f5e60fbc82ee35c1bf80eee 6bbd0854
SSHPacket: [RECV] Encrypted packet:
00000000: 59 1f 7a 88 33 10 b8 64 49 a5 e5 c1 41 e8 aa 5a Y.z.3..dI...A..Z
00000010: 7b 17 76 ea bd 94 e1 4b a3 ed 1b 6b 6d 66 98 0e {.v....K...kmf..
00000020: a2 8f d7 2a 5f 5e 60 fb c8 2e e3 5c 1b f8 0e ee ...*^.....
00000030: 6b bd 08 54 ..... k..T

SSHPacket: [RECV] Decrypted packet:
00000000: 00 00 00 1c 0a 06 00 00 00 0c 73 73 68 2d 75 73 .....ssh-us
00000010: 65 72 61 75 74 68 34 21 93 a1 5d 7c e1 08 5d ea  erauth4!..][..].

SSHClient: Validate MAC #4 on
00000000: 00 00 00 1c 0a 06 00 00 00 0c 73 73 68 2d 75 73 .....ssh-us
00000010: 65 72 61 75 74 68 34 21 93 a1 5d 7c e1 08 5d ea  erauth4!..][..].

SSHClient: mac_s2c is
00000000: f5 98 ab 55 d1 5d 04 83 f3 2d 03 c9 3d 9f 5f 0b ...U.]...=...
00000010: 33 ec c8 0f 17 41 0d 93 69 50 a0 cc e1 b8 ba 87 3....A..iP.....

SSHClient: MAC VALIDATED with offset 4
SSHSocketWrapper: Received packet[4]: 6 (SSH2_MSG_SERVICE_ACCEPT)

```

IV.III. Solicitud de Conexion¹⁵

Una vez que tenemos permisos para autenticar, podemos solicitar una conexion

```

SSHPacket: Creating packet with length 76, padding length 20 (12), payload length 55 - 16
SSHPacket: [SEND] Unencrypted packet:
00000000: 00 00 00 4c 14 32 00 00 00 07 61 6c 61 6b 72 61 ...L.2...alakra
00000010: 6e 00 00 0e 73 73 68 2d 63 6f 6e 6e 65 63 74 n....ssh-connect
00000020: 69 6f 6e 00 00 08 70 61 73 73 77 6f 72 64 00 ion...password.
00000030: 00 00 00 0e 61 6c 61 6e 72 61 6e 57 20 6c 44 .....nalmranW 10
00000040: 83 12 e7 f1 32 93 a5 79 7d f8 27 e9 d6 d3 b1 c1 ....2..y).....

SSHPacket: [SEND] Encrypted packet:
00000000: 88 e0 23 89 a6 46 21 89 86 af 85 75 3c c2 c0 f8 ...#.F1....u<...
00000010: 95 43 b5 5d b0 4f 0d 6e 2c dd 59 71 98 21 04 45 .C.).O.n.Yq.I.E
00000020: bb 90 a1 68 c2 92 58 71 ce ec 8a bc 28 58 07 c6 ...h.Xq... (X...
00000030: 5c 26 05 0e 22 63 87 24 82 27 46 2f c0 f1 be \&...c.../f...
00000040: 77 43 d2 02 53 51 bb 87 88 60 44 62 49 dd 39 31 wC.SQ...Db1.91
00000050: cf 29 5a 9c 88 e2 79 e6 78 d5 a3 6e 31 94 1c bd .Z)...y.x..n1...
00000060: e4 ee 93 93 .....

SSHSocketWrapper: c >> s [100]: 88e02389a646218986af85753cc2c0f8 9543b55db04f0d6e2cdd597198210445 bb90a168c2925871ceec8abc285807c6 5c26050622f263872f8227462fccf1be
7743d2025351bb87886046249dd3931 cf295a9c88e279e678d5a36e31941cbd e4ee9393
SSHSocketWrapper: s >> c [5]: 2b15e904eb
SSHPacket: length=12 pad=10
SSHSocketWrapper: s >> c [11]: ca618cd8d5d519da85224b
SSHSocketWrapper: s >> c [20]: 1d0fe41b35084b6c9508c08fa54ef8692a 721e3da9
SSHPacket: [RECV] Encrypted packet:
00000000: 2b 15 e9 04 eb ca 61 8c d8 d5 19 da 05 22 4b +....&....."K
00000010: 1d 0f e4 1b 35 08 4b 69 58 0c 8f a5 4e f8 69 2a ...5.KiX...N.i*
00000020: 72 1e 3d a9 ..... n..

SSHPacket: [RECV] Decrypted packet:
00000000: 00 00 00 0c 0a 34 0b 0b 5e cc 16 50 ea 2f f4 7d ....4..&..P../

SSHClient: Validate MAC #5 on
00000000: 00 00 00 0c 0a 34 0b 0b 5e cc 16 50 ea 2f f4 7d ....4..&..P../

SSHClient: mac_s2c is
00000000: f5 98 ab 55 d1 5d 04 83 f3 2d 03 c9 3d 9f 5f 0b ...U.]...=...
00000010: 33 ec c8 0f 17 41 0d 93 69 50 a0 cc e1 b8 ba 87 3....A..iP.....

SSHClient: MAC VALIDATED with offset 5
SSHSocketWrapper: Received packet[5]: 52 (SSH2_MSG_USERAUTH_SUCCESS)
SSHClient: Response: SSHPacket(12, 10, b'&...', b'\x00\x0b\x0c\x16\x0e\xa5f4'), 52 (SSH2_MSG_USERAUTH_SUCCESS))

```

Una vez que realizamos esto, si es exitosa, recibimos varios mensajes del servidor:

1. HostKeys: Autenticacion del servidor
2. Confirmacion de la apertura del canal

```

SSHPacket: [RECV] Encrypted packet:
00000000: 13 b4 e6 e3 f8 54 bc aa 42 12 c9 73 6e 93 f6 1f .....T...B...sn...
00000010: de ae 6b cf e0 f8 52 85 90 2c 93 b7 fd 71 d5 20 ...k...R.....q.
00000020: 0e 43 7a 4f 7c 3a 04 e1 f8 a1 1e 08 4c d2 08 04 .Cz0]:.....L...
00000030: 7a 29 b1 08 ..... z)..

SSHPacket: [RECV] Decrypted packet:
00000000: 00 00 00 1c 0a 5b 00 00 00 00 00 00 00 00 00 .....[.....
00000010: 00 00 00 00 80 00 ec f5 ab ee ca 94 21 b5 e2 7e .....!...~

SSHClient: Validate MAC #7 on
00000000: 00 00 00 1c 0a 5b 00 00 00 00 00 00 00 00 00 .....[.....
00000010: 00 00 00 00 80 00 ec f5 ab ee ca 94 21 b5 e2 7e .....!...~

SSHClient: mac_s2c is
00000000: f5 98 ab 55 d1 5d 04 83 f3 2d 03 c9 3d 9f 5f 0b ...U.]...=...
00000010: 33 ec c8 0f 17 41 0d 93 69 50 a0 cc e1 b8 ba 87 3....A..iP.....

SSHClient: MAC VALIDATED with offset 7
SSHSocketWrapper: Received packet[7]: 91 (SSH2_MSG_CHANNEL_OPEN_CONFIRMATION)
SSHClient: Channel open: SSHMessageChannelOpenConfirmationPacket(recipient_channel=0, sender_channel=0, initial window size=0, maximum packet size=32768)

```

Y estamos en posicion de enviar un comando

IV.IV. Ejecucion¹⁶

¹⁵RFC4252

¹⁶RFC4254

```

SSHPacket: [SEND] Unencrypted packet:
00000000: 00 00 00 3c 22 62 00 00 00 00 00 00 04 65 78    ...<"b.....ex
00000010: 65 63 01 00 00 00 07 65 63 68 6f 20 68 69 35 75    ec.....echo hi5u
00000020: f8 c7 27 65 73 6b c1 3e 35 0c 58 03 7b 93 bd 9c    ..'esk.>5.X.{...
00000030: 7a 7b cf be 0f 15 2d 7a 6a bf 4d ff 32 3a 31 b6    z{.....-zj.M.2:1.

SSHPacket: [SEND] Encrypted packet:
00000000: 01 c6 a8 de 8e 93 76 32 02 9d 57 dc 95 45 da 82    .....v2..W...E..
00000010: c4 c9 b8 3b de 32 c6 9e 8b 91 87 2b 4d d5 52 c6    ...;..2.....+M.R.
00000020: 9b 36 72 e9 eb 38 41 41 4b a3 59 6d f7 cf f2 82    .6r..8AAK.Ym....
00000030: a5 7a 59 6c 9a 82 5e 2d 87 74 7b 28 e3 94 21 72    .zYl..^-.t{(!r
00000040: 9d 60 06 5b 5a c6 27 98 98 ae e6 48 41 54 82 62    .^[Z.'...HAT.b
00000050: 8d 6a dc ad                                           .j..

```

Enviamos un comando de EXEC con payload echo hi

Y recibimos varios mensajes del servidor:

- `SSH2_MSG_CHANNEL_WINDOW_ADJUST`: Incrementando el tamaño maximo de un paquete.
- `SSH2_MSG_CHANNEL_EXTENDED_DATA`: Con informacion del path, y otras variables de entorno
- `SSH2_MSG_CHANNEL_REQUEST`: Pide abrir un canal de comunicacion hacia nosotros
- `SSH2_MSG_CHANNEL_DATA`: Optimisticamente envia la respuesta a nuestra peticion antes que aceptemos abrir el canal.

```

SSHPacket: [RECV] Encrypted packet:
00000000: be 7f 62 38 60 01 53 e3 f1 19 52 9a c6 03 a2 42 ..b8^1S...R...B
00000010: 83 b6 7a 75 95 34 cd 04 75 e0 50 15 ab 7e 35 ..zu.44...u.P...5
00000020: 94 2b 4c ed 8f a6 b7 06 33 a2 64 f6 e6 87 60 05 ..+L.....3.d...^
00000030: ed ba 7e 25 ..~%

SSHPacket: [RECV] Decrypted packet:
00000000: 00 00 00 1c 0f 5e 00 00 00 00 00 00 03 68 69 .....^.....hi
00000010: 0a ad 4c 72 48 0b a1 49 a5 00 5e 39 1e 88 0d c6 ..LrH..I...^9....

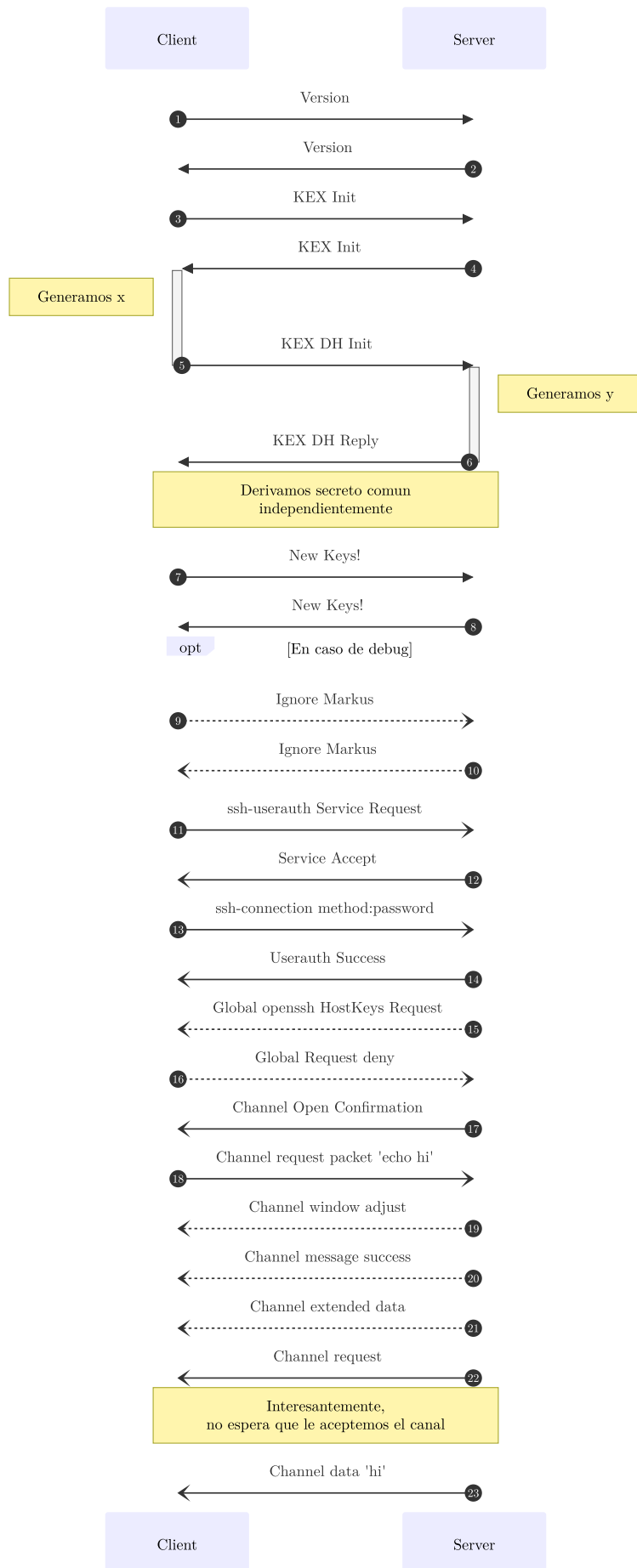
SSHClient: Validate MAC #12 on
00000000: 00 00 00 1c 0f 5e 00 00 00 00 00 00 03 68 69 .....^.....hi
00000010: 0a ad 4c 72 48 0b a1 49 a5 00 5e 39 1e 88 0d c6 ..LrH..I...^9....

SSHClient: mac_s2c is
00000000: 7f 17 1e 86 88 20 4a be 88 2d 02 75 36 bd 39 d1 .....J...-u6.9.
00000010: 84 5f 29 34 07 0c cf 1c fc 20 c5 01 a6 3c 2c f2 .._)4.....<,

SSHClient: MAC VALIDATED with offset 12
SSHSocketWrapper: Received packet[12]: 94 (SSH2_MSG_CHANNEL_DATA)
SSHClient: Response: SSHMessageChannelDataPacket(recipient_channel=0, data=b'hi\n')
SSHClient: Channel data: b'^\x00\x00\x00\x00\x00\x00\x00\x00\x03hi\n' channel data.recipient_channel=0 channel data.data=b'hi\n'

```

V. Apéndice II



Bibliografía

[1] «The story of the SSH port is 22. — ssh.com». 2017.

[2] «ietf.org». 2006.

Los RFC mencionados tienen links.