Accès et Configuration à Distance Sécurisées

Dr. Nihel Ben Youssef

10/2015

Références

T. Ylonen, RFC4250-4256 The Secure Shell (SSH), Internet Engineering Task Force (IETF),2006.

William Stallings. Cryptography and Network Security, Pearson, 6. 1998

1. Présentation de SSH

2. Transport Layer Protocol
   - Première connexion
   - Initialisation
   - Négociation
   - Echange de clés
   - Cryptage et hachage du trafic

3. Authentification du client
   - Authentification par mot de passe
   - Authentification par clé privée

Présentation de SSH

Définition

- Secure Shell (SSH) est un protocole conçu pour sécuriser l'accès aux machines distantes. En terme d'administration réseaux, SSH permet de configurer des équipements réseau tels que les routeurs ou firewalls et de gérer des services distants (serveurs).
- Le but de SSH est d'offrir une alternative au protocole d'accès à distance comme Telnet qui n'offre pas notamment la confidentialité et l'intégrité des données.
- La dernière version de SSH est SSH2. Elle est définie et standardisée dans IETF (Internet Engineering Task Force) RFC4250 à RFC4256.
- OpenSSH est l'implémentation du protocole SSH dans les systèmes GNU/Linux et BSD. Il s'agit d'une suite de logiciels (ssh, sshd, scp, sftp) développés depuis 1999 par l'équipe d'OpenBSD. Parmis les services supplémentaires offerts par SSH, on cite le transfert de fichiers et emails sécurisés.

Composition de SSH

Couches protocolaires de SSH

SSH est organisé en trois sous couches utilisant TCP comme protocole de communication.

- **Transport Layer Protocol** : permet l'authentification du serveur, la confidentialité et l'intégrité des données. cette couche fournit optionnellement la compression.
- **User Authentication Protocol** : permet d'authentifier le client auprès du serveur.
- **Connection Protocol** : permet de multiplexer plusieurs canaux de communication à travers une seule connection SSH.

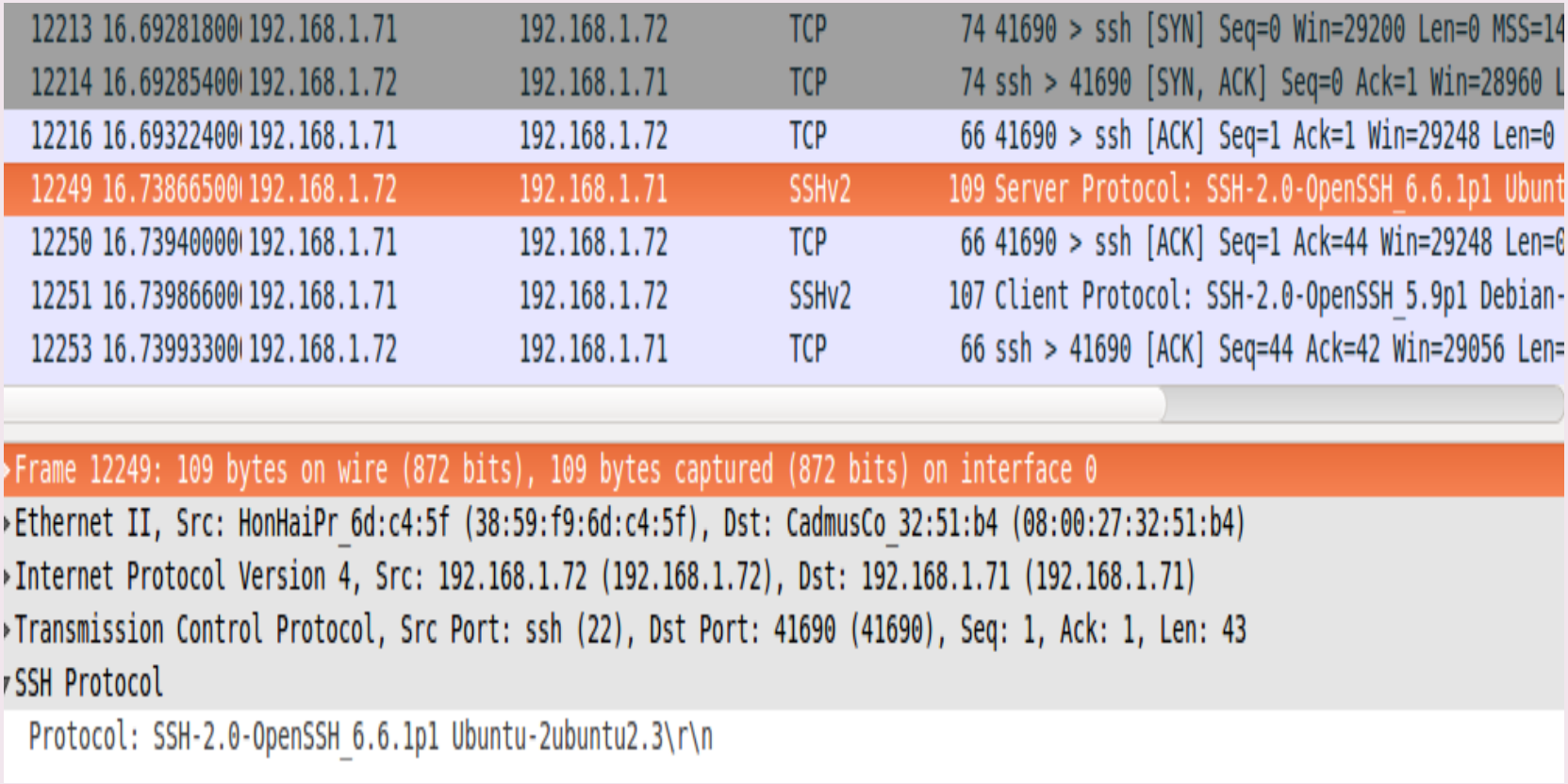
Première Connexion

A la première connexion, un client *ssh* demande à établir une connexion SSH à travers le nom d'utilisateur et le nom ou l'adresse IP de la machine distante.

```
nihelubuntu@nihelubuntu-VirtualBox:~$ ssh ahmed@192.168.1.72
The authenticity of host '192.168.1.72 (192.168.1.72)' can't be established.
ECDSA key fingerprint is 20:44:b5:e2:f5:26:41:d0:7c:a7:f9:a6:4e:ad:85:bf.
Are you sure you want to continue connecting (yes/no)? yes
```

Phase 1 : Initialisation de la session

Après la phase de connexion TCP, Les clients et serveurs échangent leurs version receptives **SSH_PROTOVERSION_SOFTWAREVERSION**.



Phase 2 : Négociation

Le client ainsi que le serveur se mettent d'accord sur les algorithmes (cryptage,MAC,compression) à utiliser dans les deux sens.

SSH_MSG_KEXINIT.

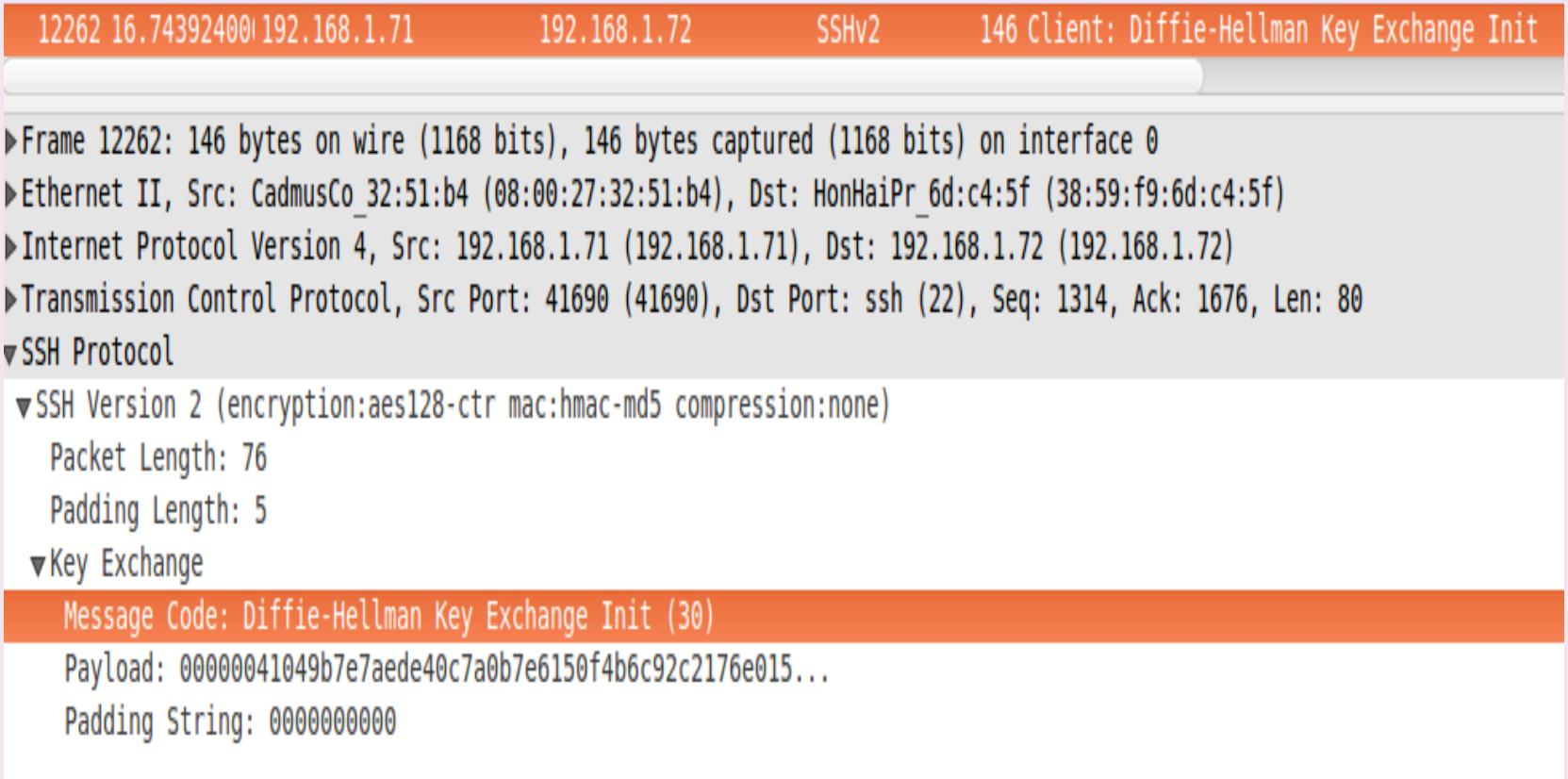
```
12254 16.74091900 192.168.1.71    192.168.1.72    SSHv2    1338 Client: Key Exchange Init

▼SSH Version 2 (encryption:aes128-ctr mac:hmac-md5 compression:none)
  Packet Length: 1268
  Padding Length: 8
 ▼Key Exchange
   Message Code: Key Exchange Init (20)
  ▼Algorithms
    Cookie: 599630b396298a50eb50f8c893bd0aa8
    kex_algorithms length: 183
    kex_algorithms string: ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,dif
    server_host_key_algorithms length: 314
    server_host_key_algorithms string [truncated]: ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@ope
    encryption_algorithms_client_to_server length: 157
    encryption_algorithms_client_to_server string: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cb
    encryption_algorithms_server_to_client length: 157
    encryption_algorithms_server_to_client string: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cb
    mac_algorithms_client_to_server length: 167
    mac_algorithms_client_to_server string: hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-sha2-256,hmac-sha2-256-96,hmac-sha2-
    mac_algorithms_server_to_client length: 167
    mac_algorithms_server_to_client string: hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-sha2-256,hmac-sha2-256-96,hmac-sha2-
    compression_algorithms_client_to_server length: 26
    compression_algorithms_client_to_server string: none,zlib@openssh.com,zlib
    compression_algorithms_server_to_client length: 26
```

Phase 3 : Echange de clé Symmétrique

- Le client et le serveur entamme le processus pour échanger d'une manière sécurisée la clé symmétrique K qui va être utilisée pour générer essentiellement les clés de chiffrement et de MAC (Message Authenticaton Code) d'une manière similaire à TLS /SSL.
- L'échange de clé est communément effectué à travers le protocole DH (Diffie Hellman) mais récemment, une alternative du protocole a vu le jour. Il s'agit de ECDH (Elliptic Curve Diffie Hellman), une version du protocole DH en utilisant la cryptographie sur les courbes elliptiques(ECC).
- Dans ce cours, nous allons uniquement traiter la première version de DH.

Phase 3 : Echange de clé Symmétrique

Le client initie l'échange Diffie-Hellman.



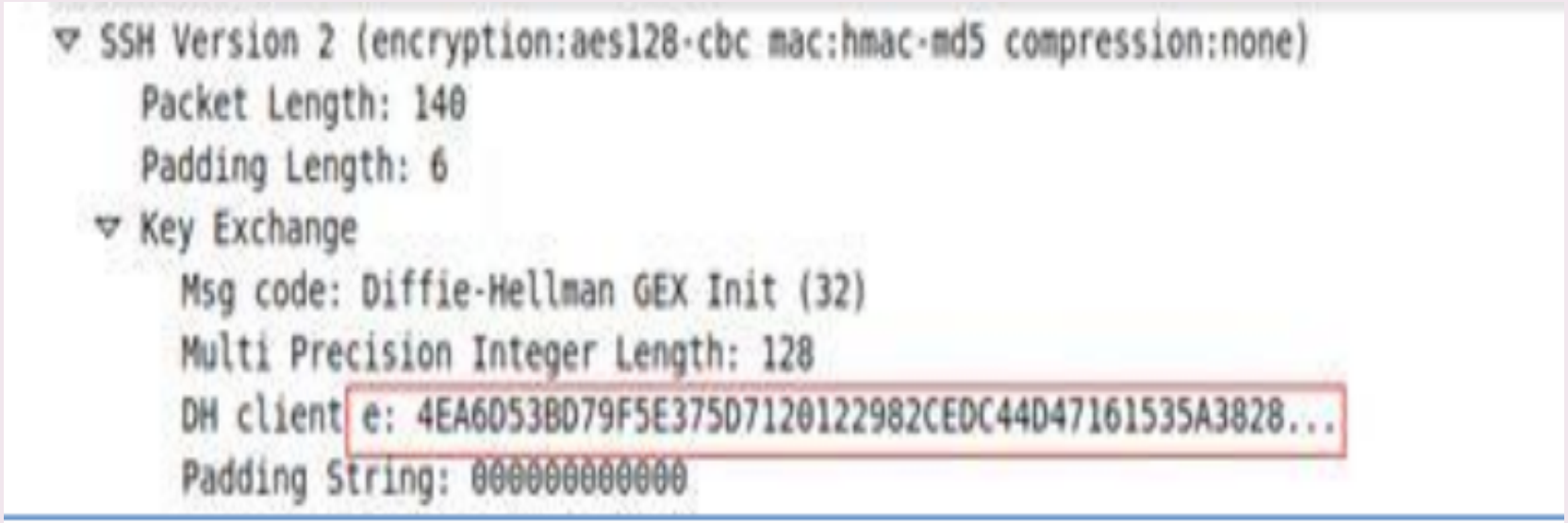
Phase 3 : Echange de clé Symmétrique

Le serveur envoie un nombre premier P très grand et la base G.

```
12278 16.76665900 192.168.1.72      192.168.1.71      SSHv2    378 Server: New Keys

▸Frame 12278: 378 bytes on wire (3024 bits), 378 bytes captured (3024 bits) on interface 0
▸Ethernet II, Src: HonHaiPr_6d:c4:5f (38:59:f9:6d:c4:5f), Dst: CadmusCo_32:51:b4 (08:00:27:32:51:b4)
▸Internet Protocol Version 4, Src: 192.168.1.72 (192.168.1.72), Dst: 192.168.1.71 (192.168.1.71)
▸Transmission Control Protocol, Src Port: ssh (22), Dst Port: 41690 (41690), Seq: 1676, Ack: 1394, Len: 312
▾SSH Protocol
 ▾SSH Version 2 (encryption:aes128-ctr mac:hmac-md5 compression:none)
   Packet Length: 292
   Padding Length: 11
  ▾Key Exchange
    Message Code: Diffie-Hellman Key Exchange Reply (31)
    Multi Precision Integer Length: 104
    DH modulus (P): 0000001365636473612d736861322d6e6973747032353600...
    Multi Precision Integer Length: 65
    DH base (G): 04cf6f6e111976c11ffd7c42a49cc8a00d7ecd26eb120b3d...
    Payload: 000000620000001365636473612d736861322d6e69737470...
    Padding String: 00000000000000000000
```

Phase 3 : Echange de clé Symmétrique

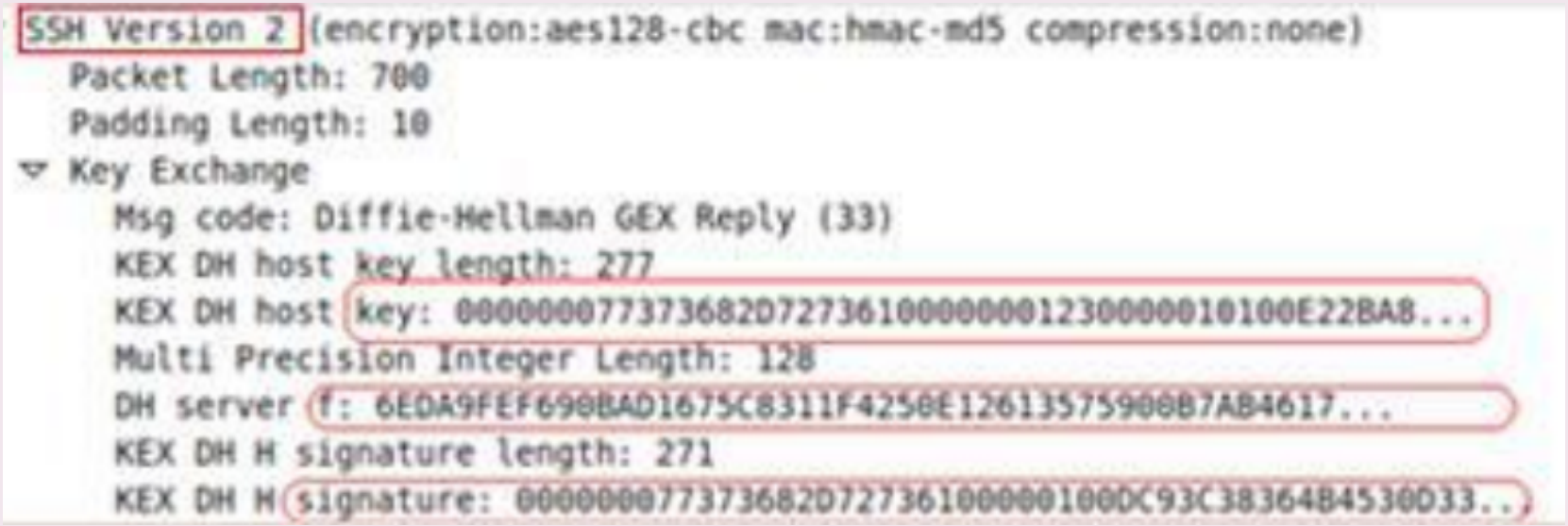
Le client choisit un nombre aléatoire privé Cx et calcule puis envoi : **e**= G^{Cx} **mod P**(DIFFIE_ HELLMAN_ GEX_ INIT).



Phase 3 : Echange de clé Symmétrique

- Le serveur choisit un nombre aléatoire privé Sx.
- Il calcule : **f**= G^{Sx} **mod P**.
- Il calcule la clé symmétrique : **k**= e^{Sx} **mod P**.
- Il calcule h=hash(Vc|| Vs || Algoc || Algos ||KpubS || e || f || K). La fonction hash étant négociée.
- *Vc et Vs étant les versions respectives du client et du serveur.*
- *Algoc et Algos étant la liste des algorithmes envoyés par les deux parties dans la phase de négociation (**SSH_MSG_KEXINIT**).*
- *KpubS est la plé publique du serveur.*
- Le serveur effectue une signature s sur h.
- Au final, il envoie (KpubS || f || s) :(**DIFFIE_ HELLMAN_ GEX_ REPLY**).

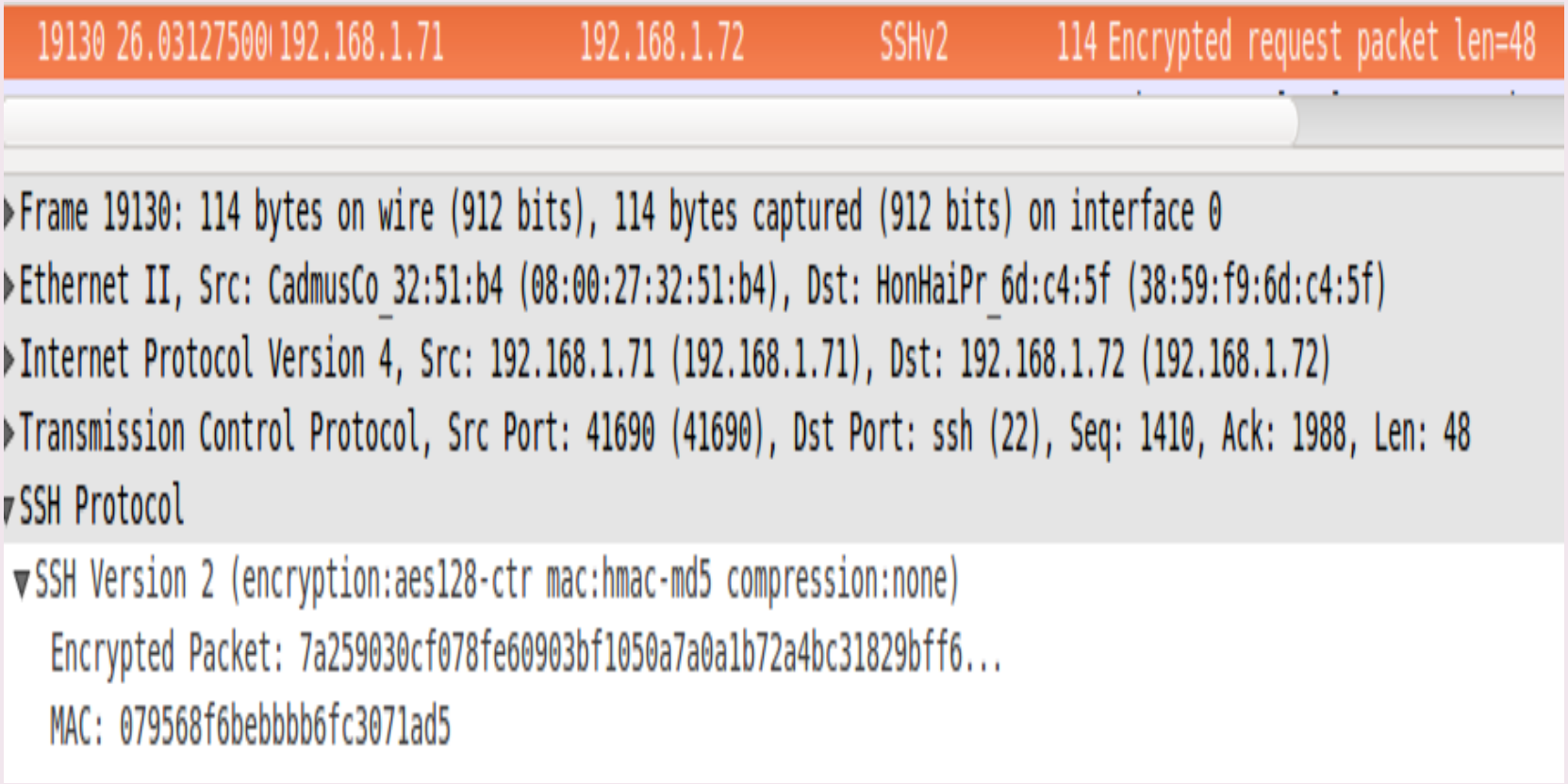
Phase 3 : Echange de clé Symmétrique



Phase 4 : Cryptage et hachage du trafic

Avant de commencer l'envoi de paquets chiffrés, le client doit vérifier l'authenticité de la clé publique Kpubs à travers son empreinte donnée en première connexion.
Une fois validée, la machine serveur est reconnue et sa clé est stockée dans une base locale **known_hosts**. Une fois la clé K calculée au niveau du client et du serveur. Elle est utilisé pour générer plusieurs clés comme par analogie dans le protocole SSL. Les clés générées sont ClientIV, ServerIV, ClientMAC, ServerMAC et ClientCipher et ServerCipher.

Phase 4 : Cryptage et hachage du trafic



Rappels sur les clefs généréesà partir de K (Analogie TLS/SSL)

Source : mon cours TLS/SSL

Client Cipher : utilisée pour chiffrer les données du client vers le serveur.

Server Cipher: utilisée pour chiffrer les données du serveur vers le client.

Client MAC : utilisée dans la fonction cryptographique de hachage **HMAC** coté client pour le contrôle d'intégrité.

Server MAC : utilisée dans la fonction cryptographique de hachage **HMAC** par le serveur pour le contrôle d'intégrité.

Client IV : Vecteur d'initialisation utilisé par le client au niveau du **mode CBC** lors du chiffrement symétrique des données.

Server IV : Vecteur d'initialisation utilisé par le serveur au niveau du **mode CBC** lors du chiffrement symétrique des données.

Rappels sur la fonction HMAC(Analogie TLS/SSL)

Source : Wikipédia

$$\mathrm{HMAC}_K(m) = h\Big((K \oplus opad) \,||\, h\big((K \oplus ipad) \,||\, m\big)\Big)$$

avec :

h : une fonction de hachage itérative(MD5,SHA),

K : la clé secrète complétée avec des zéros pour qu'elle atteigne la taille de bloc de la fonction h. Dans notre cas, il s'agit soit de la clé client MAC ou server MAC

m : le message à authentifier,

"||" désigne une concaténation,

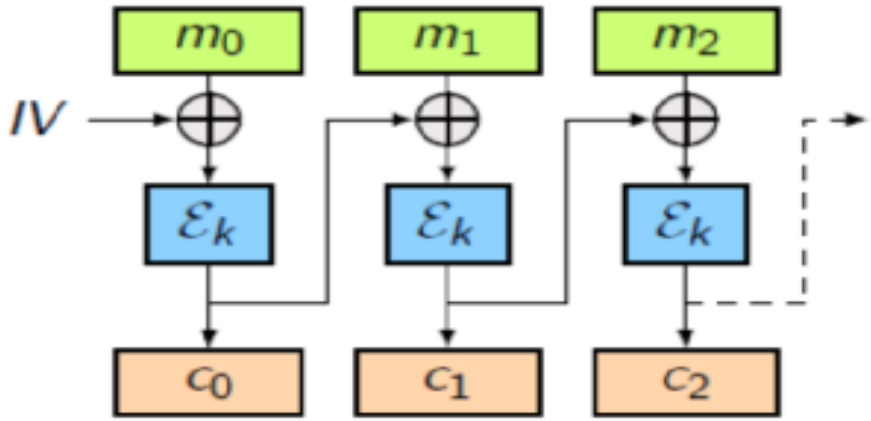
ipad et opad, chacune de la taille d'un bloc, sont définies par : ipad = 0x363636...3636 et opad = 0x5c5c5c...5c5c. Donc, si la taille de bloc de la fonction de hachage est 512, ipad et opad sont 64 répétitions des octets, respectivement, 0x36 et 0x5c. »

Rappels sur le mode symétrique CBC(Analogie TLS/SSL)

Source : Wikipédia

-Le message m à chiffrer est découpé en block (m0,m1..). Un bloc dépend de tous les précédents . Mode randomisé par la présence d'une valeur aléatoire initiale IV. Dans notre cas, il sagit soit du client IV ou Server IV.

-E_k est un algorithme de chiffrement symétrique au choix (DES,3DES,AES...)



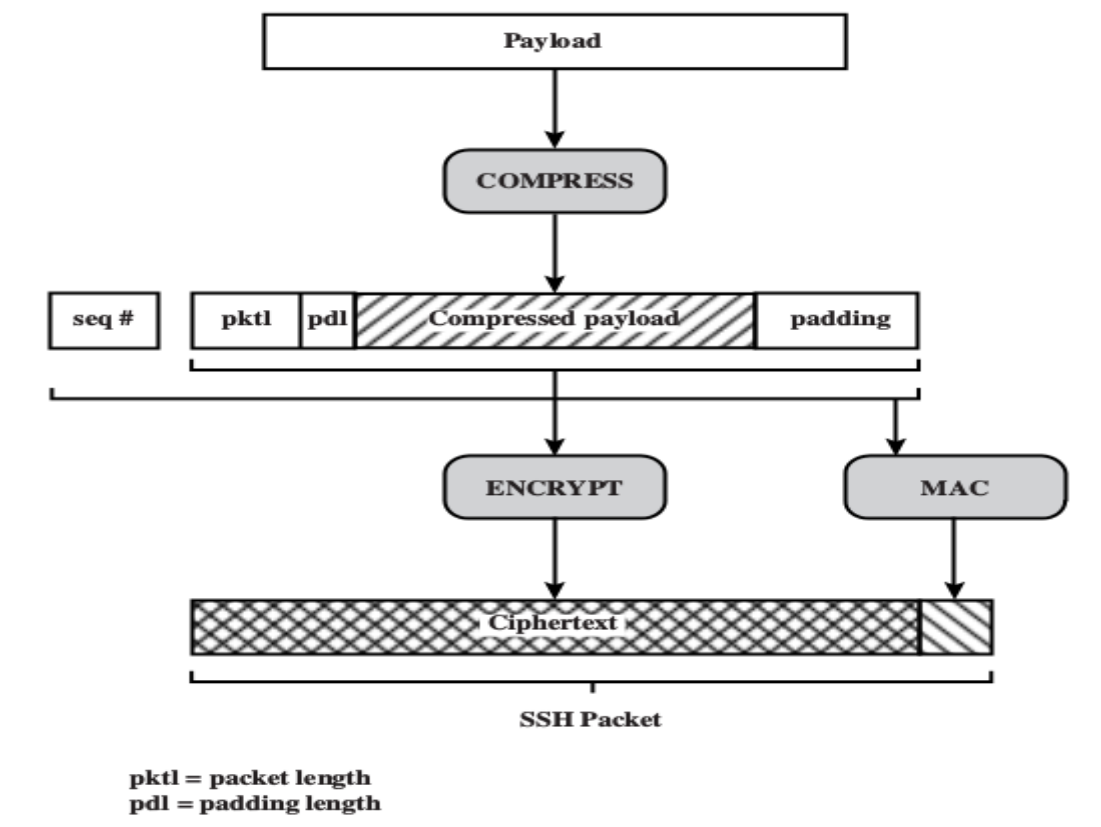
Transport Layer Protocol

Echange Sécurisé des données

- Le message ou payload d'un paquet envoyé par le client ou le serveur va être compressé selon l'algorithme négocié au départ.
- En suite, il sera crypté en utilisant un algorithme symmétrique commun avec des clés de chiffrement ClientCipher ou ServerCipher et en utilsant aussi soit ClientIV ou respectivement ServerIV si un mode utilisant les vecteurs d'initialisation comme CBC est employé.
- Finalement, un MAC du payload compressé ainsi que le numéro du séquence du paquet sera ajouté en utilisant respectivement ClientMAC ou ServerMAC.

Echange Sécurisé des donnée

Source : William Stallings. Cryptography and Network Security. Pearson 1998



Authentification par défaut

Mot de passe

La phase suivante est l'authentification du client. Par défaut, Le client doit fournir un mot de passe qui va être envoyé dans un packet crypté.

```
nihelubuntu@nihelubuntu-VirtualBox:~$ ssh ahmed@192.168.1.72
The authenticity of host '192.168.1.72 (192.168.1.72)' can't be established.
ECDSA key fingerprint is 20:44:b5:e2:f5:26:41:d0:7c:a7:f9:a6:4e:ad:85:bf.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.72' (ECDSA) to the list of known hosts.
ahmed@192.168.1.72's password:
```

Authentification par clé privée

Autre Authentification

Le client peut aussi être authentifié auprès du serveur en envoyant sa clé publique Kpubc à travers par exemple la commande **scp** qui permet de copier la clé sur le serveur distant en utilisant SSH.
Le client crypte une information connue par le serveur, par exemple la clé publique du client, par sa clé privée.
Une fois le serveur a bien vérifié que le challenge est réussi, le client est authentifié.