www.enst.fr./~urien/cours_tunnels_2024_IMT.pdf

Quelques Tunnels Sécurisés 2024

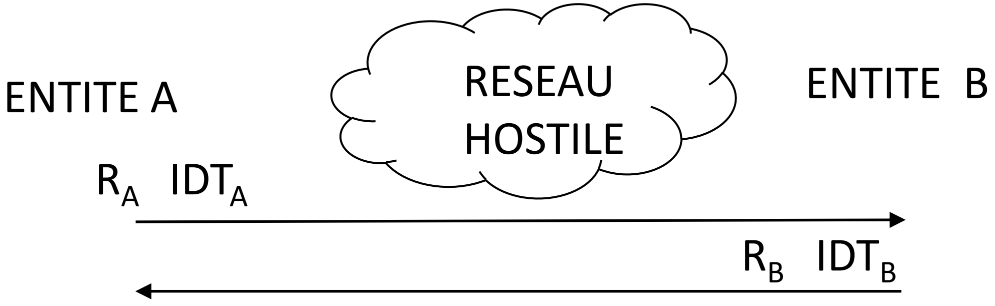
PPTP, SSH, IPSEC, TLS/DTLS

Pascal.Urien@Telecom-Paris.fr

TELECOM Paris

IP PARIS

Canal Sécurisé



$K_M = PRF(R_A, R_B, K_{S,} [IDT_A, IDT_B])$ $K_C, K_I = KDF(R_A, R_B, K_M)$

Canal de Transmission Sécurisé

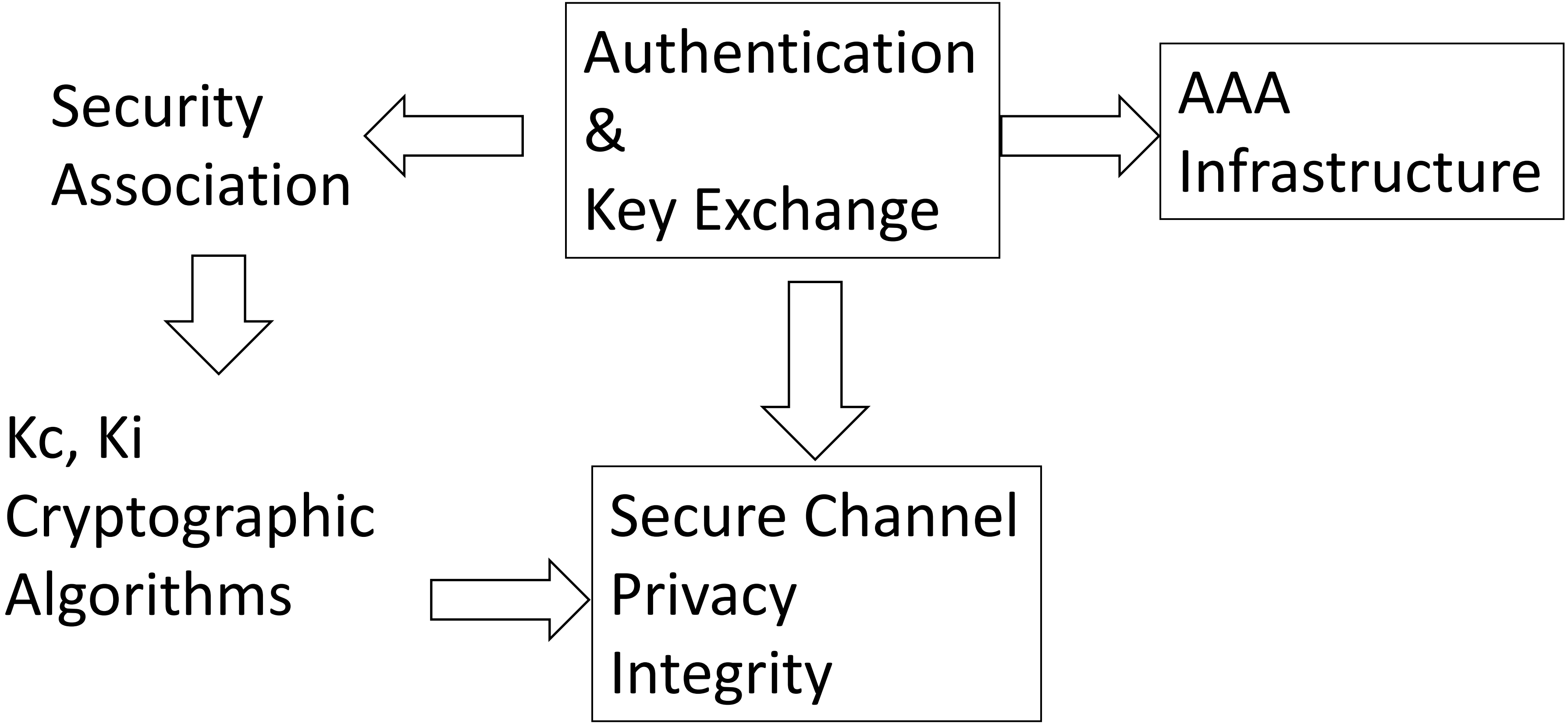
$\{M\}K_C$ $HMAC(M, K_I)$

$\{M\}K_C$ $HMAC(\{M\}, K_I)$

Canal Sécursé

- La procédure d'authentification d'une paire d'entités informatiques, parfois dénommée phase d'autorisation, consiste typiquement à échanger les identités (IDTA et IDTB) d'un couple d'interlocuteurs (appelés client/serveur ou initiateur/répondeur), deux nombres aléatoires (RA, RB) formant un identifiant unique de la session, puis d'effectuer un calcul.
- Ce dernier produit, à l'aide d'une valeur secrète (KS) un secret maître (KM), à partir duquel on déduit des clés de chiffrement (KC) et d'intégrité (KI) permettant de créer un canal sécurisé.
- Dans un contexte de cryptographie symétrique la clé KS est distribuée manuellement ; dans un contexte de cryptographie asymétrique la clé KS sera par exemple générée par A, mais chiffrée par la clé publique (e,n) de B (Ks^e mod n).
- La protection de l'identité est une préoccupation croissante avec l'émergence des technologies sans fil. Il existe divers mécanismes permettant d'obtenir cette propriété avec des degrés de confiance divers, par exemple grâce à la mise en œuvre de pseudonymes (tel que le TIMSI du GSM), du protocole de Diffie-Hellman, ou du chiffrement de certificats par la clé publique du serveur.

Mécanismes de base



Création d’un secret partagé Diffie-Hellman

- Un générateur g de Z*/nZ (avec n premier) est tel que
  - $\forall$x#0 $\exists$i g^i = x (mod n).
- Il existe φ(n-1) solutions.
- Exemple n=5, g=3
  - g^1 =3, g^2 =4, g^3 =2, g^4 =1
- x est une clé privé, g^x est une clé publique
- Un échange DH permet de construire dynamiquement un secret partagé Ks, $Ks=g^{xy} = (g^x)^y = (g^y)^x$
- Hugo Krawczyk a introduit la notion de *Randomness Extractor* (XTR)
  - *Source Key Material*, SKM= g^{xy}
  - L’entropie de SKM ($\log_2$ 1/p(SKM)) n’est pas forcément constante
  - XTR= PRF(RA|RB, SKM) = HMAC(RA | RB, SKM)
  - L’entropie de XTR est proche d’une constante

PRF et KDF, selon NIST Special Publication 800-108

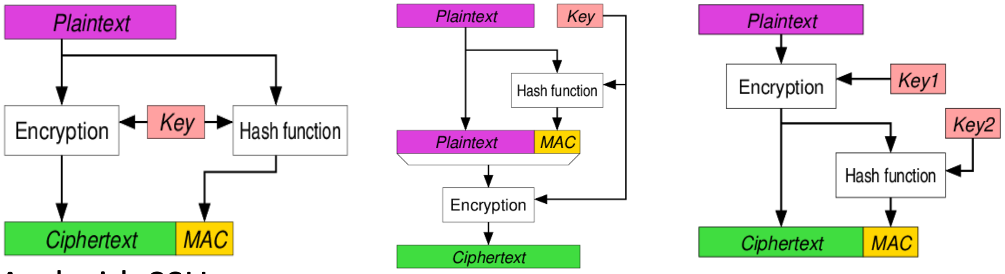
- PRF
  - Une procédure qui génère une suite d'octets pseudo aléatoire de longueur k bits.
  - PRF(s, x)
    - HMAC(key,x), CMAC(key,x)
- Mode KDF compteur
  - i compris entre 1 et L/k
  - *KDF(i) = PRF (K, i | Label | 0x00 | Context | L)*
- Mode KDF feedback
  - i compris entre 1 et L/k
  - *K(i) = PRF (K, K(i-1) | i | Label | 0x00 | Context | L)*

Création d'un secret partagé RSA

- C'est par exemple la procédure mise en œuvre par SSL
- Création d'un secret, PMS (Pre-Master-Secret)
- Calcul d'un Master Secret (MS)
  – MS = KDF(PMS, RA | RB | Label)
- Calcul des clés Kc, Ki
  – Clés = KDF (MS, RA | RB | Label)

Authenticated Encryption

- Un schéma assurant à la fois intégrité et confidentialité garantit la sécurité contre les attaques à chiffrés choisis :
  - IND-CPA + INT-CTXT ⇒ IND-CCA
    - IND-CPA= Chosen Plaintext Attack
    - INT-CTXT= Integrity of CipherTeXT
    - IND-CCA= Chosen-Ciphertext Attack



- MAC-And-Encrypt, non sûr
  - E(M) || MAC(M), exemple Keystore Android, SSH
- MAC-Then-Encrypt, mode non génériquement sûr, mais en pratique, on peut construire des schémas sûrs avec ce principe
  - E(M || MAC(M)), exemple TLS 1.0/1.1/1.2
  - **Exemple AES-CCM**
- **Encrypt-Then-MAC**, si le mode de chiffrement est « sûr » et si le MAC assure l'intégrité alors cette composition est « sûre »
  - E(M) || MAC(E(M)), exemple IPSEC

http://www.di.ens.fr/~fouque/mpri/MAC.pdf

Echange DH

- Considérons un échange de Diffie Hellman (DH) dans Z/pZ*, avec p premier, soit $(g^x)^y$ mod p, g^x est une clé publique (p_k), y une clé privée, et g un générateur d’ordre p-1.
- D’après le théorème de Sylow, si l’ordre N d’un groupe se décompose en facteurs premier (qi), N= ∏ qi^{ki}, il existe des sous groupes d’ordre q^{ki}. La décomposition en facteurs premier de N= p-1 nous montre donc l’existence de sous groupes d’ordre q^{ki}. De surcroit les sous groupes d’un groupe cyclique (*monogène*) sont cycliques.
- Dans Z/pZ* il existe des générateurs d’ordre p-1, mais aussi des générateurs pour des sous groupes plus petits dont l’ordre divise p-1. Une attaque possible dans les échanges DH est de proposer une clé publique p_k= $g_d{}^x$ utilisant un générateur dont l’ordre d divise p-1 (d | p-1).

Générateurs

- Il existe φ(p-1) générateurs (φ étant le nombre d'Euler), le nombre maximum de générateurs est (p-1)/2, c'est à dire le nombre d'entiers impairs inférieurs à p. Si p est de la forme $p=1+2^n$, $\varphi(p-1)= \varphi(2^n)= 2^{n-1}= (p-1)/2$.
- Une méthode consiste à trouver des générateurs g_{ki} d'ordre qi^{ki} (il existe $\varphi(qi^{ki}) = (qi-1).q^{ki-1}$ générateurs d'ordre qi^{ki}) puis d'en réaliser le produit. Il existe φ(p-1) générateurs soit $\varphi(p-1)= \prod \varphi(qi^{ki})= \prod (qi-1).qi^{(ki-1)}$.

Safe Prime

- Un *Safe Prime* p est de la forme p=2q + 1 avec q premier, q est le *Sophie Germain* prime de p. On en déduit p-1 = 2q , et φ(p-1)= q-1. Il existe un générateur d'ordre 2 (p-1, puisque $(p-1)^2=1$ mod p) et q-1 générateurs d'ordre q.
- Si p = 7 mod 8 (voir RFC 7919), p est un diviseur du nombre de Mersenne Mq = 2^q-1, on en déduit que 2 est un générateur d'ordre q. Par la suite 2^k avec k є [1, q-1] est un générateur d'ordre q.
- Les générateurs d'ordre p-1 sont obtenus par les produits (p-1) 2^k mod p.

Attaque et Contre-Mesure

- Considérons un groupe d'ordre N= $\prod qi^{ki}$. H un sous groupe d'ordre q premier (q|n), g_q est un générateur de H d'ordre q= qi; par exemple le plus grand qi.
- Soit d un diviseur de N, g_d un générateur d'ordre d < q.
- Par exemple dans Z/pZ d | (p-1).
- $p_k = g_d^x$ x ϵ [1, d] une clé publique malicieuse, $(g_d^x)^d = 1$, il existe d secrets partagés DH = g_d^{xy} y ϵ [1, q]
- $p_k = g_q^x$ une clé publique bien formée, c'est à dire un générateur de H d'ordre q, $(g_q^x)^q = 1$ il existe q secrets partagés DH.
- Le test p_k^q =1 (implémenté par exemple dans OPENSSL) identifie une clé publique bien formée d'ordre q. Une clé publique malicieuse dont l'ordre d ne divise pas q est détectée par ce test.

PPTP
Point to Point Protocol

Flag 0x7E	Address 0xFF	Control 03	Protocol 2 bytes	information 1500 octets max	CRC 2 bytes	Flag 0x7E

- PPP (RFC 1661, 1994) est un protocole très utilisé par les MODEMs et les accès DSL
- La trame PPP utilise le format HDLC (ISO 3309).
- L'authentification PPP est réalisée avant l'allocation d'une adresse IP

- Protocol Field Value
  - 0x0021 : IP
  - 0xC021 : Link Control Protocol (LCP)
  - 0x8021 : Network Control Protocol (NCP)
  - 0xC023 : Password Authentication Protocol (PAP)
  - 0xC025 : Link Quality Report (LQR)
  - 0xC223 : Challenge Handshake Authentication Protocol (CHAP)

```
+-------+        +------------+      +---------------+
|       |  UP    |            | OPENED      |               | SUCCESS/NONE
| Dead  |------->| Establish  |----------->| Authenticate  |--+
|       |        |    LCP     |      | Auth.Protocol |  |
+-------+        +------------+      +---------------+  |
   ^                  |                    |             |
   |            FAIL  |              FAIL  |             |
   +<-----------------+         +----------+             |
   |                            |                        |
   |             +------------+ |         +----------+   |
   |        DOWN |            | |  CLOSING |          |   |
   +-------------| Terminate  |<---+<----------| Network  |<--+
                 |            |              |   NCP    |
                 +------------+              +----------+
```

Au sujet de PPP

PPP Authentication 1/2

LCP Coding

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
LCP (code), 1-Request 2-Ack 3-Nak 4-Reject C-IDENTITY
```

LCP Option=3, Authentication Request

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type=3    |    Length=5   | Authentication-Protocol= c223 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Algorithm=5 MD5|
+-+-+-+-+-+-+-+-+
```

CHAP coding

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
```

Code

1- Challenge, 2-Response

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Value-Size   |  Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Name ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3-Success, 4-Failure

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Message  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

PPP Authentication 2/2

Client | Server

LCP request
Authentication Request CHAP (Server → Client)

```
0000  20 52 45 43 56 02 20 52 45 43 56 02 c0 21 01 a5   RECV. RECV..!..
0010  00 19 02 06 00 0a 00 00 03 05 c2 23 05 05 06 eb   ...........#....
0020  46 f0 b0 07 02 08 02                              F......
```

LCP response
Authentication CHAP (Client → Server)

```
0000  20 53 45 4e 44 02 20 53 45 4e 44 02 c0 21 02 a5   SEND. SEND..!..
0010  00 19 02 06 00 0a 00 00 03 05 c2 23 05 05 06 eb   ...........#....
0020  46 f0 b0 07 02 08 02                              F......
```

LCP identity (Client → Server)

```
0000  20 53 45 4e 44 02 20 53 45 4e 44 02 c0 21 0c 03   SEND. SEND..!..
0010  00 12 1b 84 70 8a 4d 53 52 41 53 56 35 2e 31 30   ....p.MSRASV5.10
```

LCP identity (Client → Server)

```
0000  20 53 45 4e 44 02 20 53 45 4e 44 02 c0 21 0c 04   SEND. SEND..!..
0010  00 1b 1b 84 70 8a 4d 53 52 41 53 2d 30 2d 50 41   ....p.MSRAS-0-PA
0020  53 43 41 4c 55 52 49 45 4e                       SCALURIEN
```

CHAP Challenge (Server → Client)

```
0000  20 52 45 43 56 02 20 52 45 43 56 02 c2 23 01 7a   RECV. RECV..#.z
0010  00 1d 10 45 41 9a 49 11 79 e2 c4 f1 86 5d 3a c7   ...EA.I.y....]:.
0020  fb 66 e1 73 74 61 63 6b 73 74 67                 .f.stackstg
```

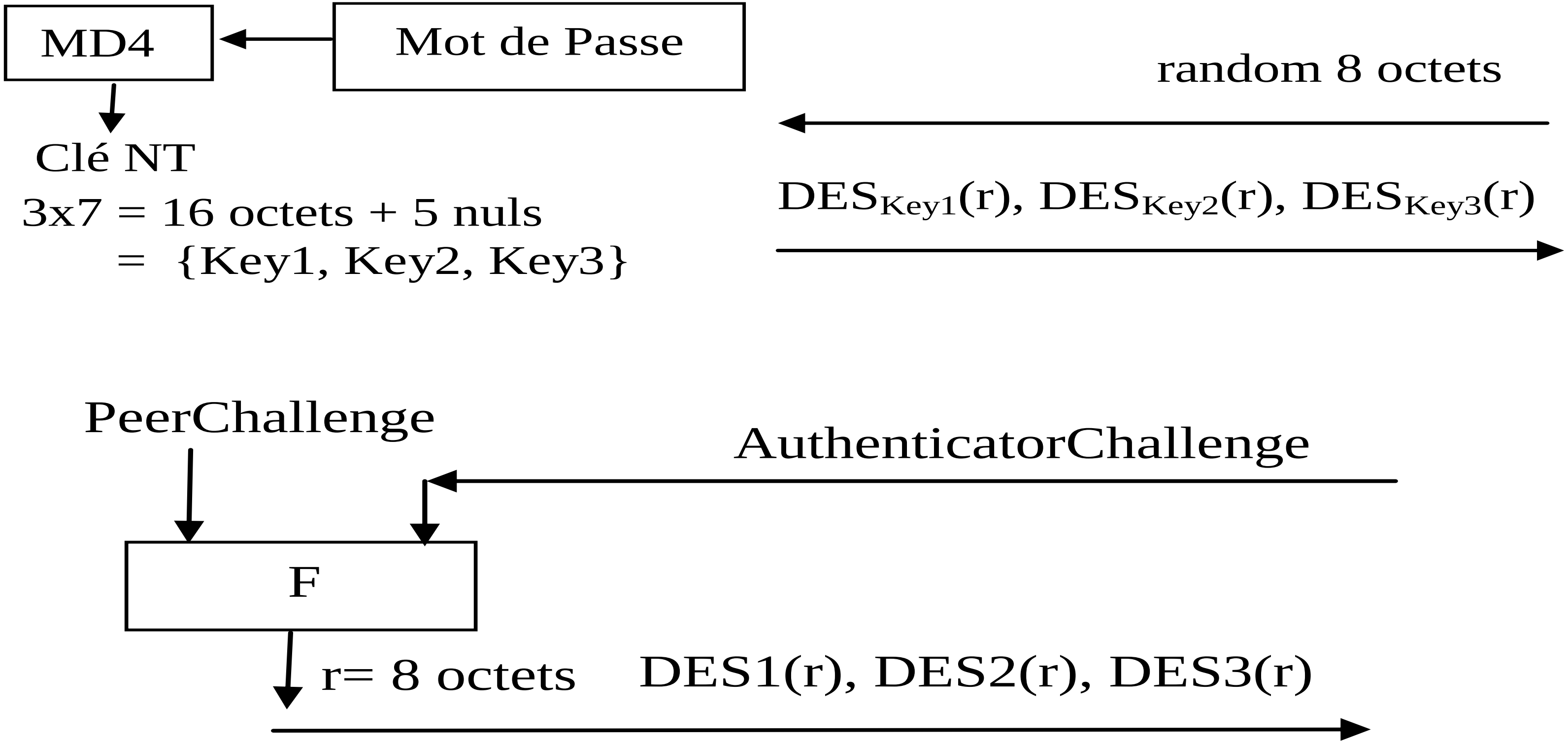
CHAP Response (Client → Server)

```
0000  20 53 45 4e 44 02 20 53 45 4e 44 02 c2 23 02 7a   SEND. SEND..#.z
0010  00 26 10 6b b6 65 8e 71 a5 39 23 a0 fe 30 45 b5   .&.k.e.q.9#..0E.
0020  81 df f9 70 75 72 69 30 30 30 33 40 6c 73 75 72   ...puri0003@lsur
0030  66 2e 66 72                                       f.fr
```

CHAP Success (Server → Client)

```
0000  20 52 45 43 56 02 20 52 45 43 56 02 c2 23 03 7a   RECV. RECV..#.z
0010  00 04                                             ..
```

Clé NT, MSCHAPv1, MSCHAPv2



PPTP Point to Point Tunneling Protocol

- PPTP: utilise le port serveur 1723
- Il établit une connexion TCP entre un client PPTP et un serveur PPTP
- PPTP permet d'ouvrir plusieurs sessions PPP, optionnellement chiffrées selon le protocole MPPE (Microsoft Point To Point Encryption).
  - Chaque session PPP est encapsulée par un entête GRE (*Generic Routing Encapsulation*) et identifiée par un attribut CallID

PPTP: principaux messages

- Start-Control-Connection-Request
  - Ce message est envoyé par le client PPTP pour établir une connexion de contrôle PPTP.
- Start-Control-Connection-Reply
  - Réponse au message Start-Control-Connection-Request.
- Set-Link-Info
  - Ce message émis par le client PPTP ou le serveur PPTP pour fixer les options PPP négociées.

PPTP: principaux messages

- Call-Clear-Request
  - Envoyé par le client PPTP pour la fermeture d’un tunnel PPP.
- Call-Disconnect-Notify
  - Fermeture d’un tunnel PPP par le serveur PPTP en réponse à Call-Clear-Request ou pour une autre raison
- Stop-Control-Connection-Request
  - Notification par le client PPTP ou le serveur PPTP de la fermeture d’une session PPTP.
- Stop-Control-Connection-Reply
  - Réponse au message Stop-Control-Connection-Request

PPTP: principaux messages

- Outgoing-Call-Request
  - Envoyé par le client PPTP pour créer un tunnel PPTP.
  - Comporte un identifiant d'appel (*Call ID*) qui est utilisé dans l'en-tête GRE pour identifier le trafic d'un tunnel spécifique.
- Outgoing-Call-Reply
  - Réponse du serveur PPTP au message Outgoing-Call-Request
  - Comporte *Call ID* et un attribut *Peer Call ID (Call ID du request)*

Outgoing-Call-Request

```
119 19.989371000 192.168.2.33 137.194.4.241 PPTP 222 Outgoing-Call-Request
⊞ Frame 119: 222 bytes on wire (1776 bits), 222 bytes captured (1776 bits) on interface 0
⊞ Ethernet II, Src: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54), Dst: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76)
⊞ Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 137.194.4.241 (137.194.4.241)
⊞ Transmission Control Protocol, Src Port: 52253 (52253), Dst Port: 1723 (1723), Seq: 157, Ack: 157, Len: 168
⊟ Point-to-Point Tunnelling Protocol
    Length: 168
    Message type: Control Message (1)
    Magic Cookie: 0x1a2b3c4d (correct)
    Control Message Type: Outgoing-Call-Request (7)
    Reserved: 0000
    Call ID: 28781
    Call Serial Number: 2
    Minimum BPS: 300
    Maximum BPS: 100000000
    Bearer Type: Either access supported (3)
    Framing Type: Either Framing supported (3)
    Packet Receive Window Size: 64
    Packet Processing Delay: 0
    Phone Number Length: 0
    Reserved: 0000
    Phone Number:
    Subaddress:
```

Client

Outgoing Call Request


```
0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 d0 4b 88 40 00 80 06  5d 23 c0 a8 02 21 89 c2   ..K.@... ]#...!..
0020  04 f1 cc 1d 06 bb 18 74  73 c9 56 79 dc fc 50 18   .......t s.Vy..P.
0030  10 dd 0a 3f 00 00 00 a8  00 01 1a 2b 3c 4d 00 07   ...?.... ...+<M..
0040  00 00 70 6d 00 02 00 00  01 2c 05 f5 e1 00 00 00   ..pm.... .,......
0050  00 03 00 00 00 03 00 40  00 00 00 00 00 00 00 00   .......@ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00
```

Outgoing-Call-Reply

No.	Time	Source	Destination	Protocol	Length	Info
115	19.939891000	192.168.2.33	137.194.4.241	PPTP	210	Start-Control-Connection-Request
118	19.989120000	137.194.4.241	192.168.2.33	PPTP	210	Start-Control-Connection-Reply
119	19.989371000	192.168.2.33	137.194.4.241	PPTP	222	Outgoing-Call-Request
121	20.034242000	137.194.4.241	192.168.2.33	PPTP	86	Outgoing-Call-Reply
124	20.043042000	192.168.2.33	137.194.4.241	PPTP	78	Set-Link-Info
136	20.203626000	192.168.2.33	137.194.4.241	PPTP	78	Set-Link-Info
1738	79.938646000	192.168.2.33	137.194.4.241	PPTP	70	Echo-Request
1739	79.980264000	137.194.4.241	192.168.2.33	PPTP	74	Echo-Reply
2508	135.511235000	192.168.2.33	137.194.4.241	PPTP	78	Set-Link-Info
2512	135.653706000	192.168.2.33	137.194.4.241	PPTP	70	Call-Clear-Request

```
⊞ Frame 121: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
⊞ Ethernet II, Src: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76), Dst: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54)
⊞ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
⊞ Transmission Control Protocol, Src Port: 1723 (1723), Dst Port: 52253 (52253), Seq: 157, Ack: 325, Len: 32
⊟ Point-to-Point Tunnelling Protocol
    Length: 32
    Message type: Control Message (1)
    Magic Cookie: 0x1a2b3c4d (correct)
    Control Message Type: Outgoing-Call-Reply (8)
    Reserved: 0000
    Call ID: 34176
    Peer Call ID: 28781
    Result Code: Connected (1)
    Error Code: None (0)
    Cause Code: 0
    Connect Speed: 100000000
    Packet Receive Window Size: 64
    Packet Processing Delay: 0
    Physical Channel ID: 0
```

= Client Call ID

Serveur

Outgoing Call Response

```
0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 48 ac 2c 40 00 34 06  49 07 89 c2 04 f1 c0 a8   .H.,@.4. I.......
0020  02 21 06 bb cc 1d 56 79  dc fc 18 74 74 71 50 18   .!....Vy ...ttqP.
0030  20 9d 75 99 00 00 00 20  00 01 1a 2b 3c 4d 00 08    .u.... ...+<M..
0040  00 00 85 80 70 6d 01 00  00 00 05 f5 e1 00 00 40   ....pm.. .......@
0050  00 00 00 00 00 00                                  ......
```

RFC 2890, Generic Routing Encapsulation (GRE)

- GRE est un protocole d’encapsulation au dessus de IP (PTCOL=47=0x2F)
- L’identifiant du protocole encapsulé est 0x880B pour PPP

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C| |K|S| Reserved0       | Ver |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Checksum (optional)      |       Reserved1 (Optional)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Key (optional)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Sequence Number (Optional)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

K: Key Present, S: sequence Present

Enhanced GRE (PPTP)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|R|K|S|s|Recur|A| Flags | Ver |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Key (HW) Payload Length    |       Key (LW) Call ID        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Sequence Number (Optional)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Acknowledgment Number (Optional)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

MPPE: Microsoft Point to Point Encryption – RFC 3078

- L'option MPPE est négociée via le protocole CCP Configuration (Compression Control Protocol de PPP), et l'option 18 .
- Le PID (PPP) du protocole MPPE est 0x00FD
- Les paquets PPP sont chiffrés par l'algorithme RC4
  - 00FD [paquet PPP Chiffré]
  - MPPE chiffre les protocoles PPP dont les identifiants sont dans l'intervalle [0x0021, 0x00FA]
- MPPE ne peut s'appliquer que dans l'état *Opened* du *CCP Control Protocol*.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type=18  |    Length=6   |         Supported Bits        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Supported Bits         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 3                   2                   1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             |H|                         |M|S|L|D|         |C|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-, -+-+-+-+
```

128-bit encryption ('S' bit set), 56-bit encryption ('M' bit set), 40-bit encryption ('L' bit set)

MPPE: Microsoft Point to Point Encryption

Serveur

```
155 20.300900000 137.194.4.241 192.168.2.33 PPP CCP 62 Configuration Request
⊟ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 48
    Identification: 0xac3c (44092)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 52
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x88e6 [validation disabled]
    Source: 137.194.4.241 (137.194.4.241)
    Destination: 192.168.2.33 (192.168.2.33)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3081
    Protocol Type: PPP (0x880b)
    Payload Length: 12
    Call ID: 28781
    Sequence Number: 12
    Acknowledgment Number: 10
⊟ Point-to-Point Protocol
    Protocol: Compression Control Protocol (0x80fd)
⊟ PPP Compression Control Protocol
    Code: Configuration Request (1)
    Identifier: 2 (0x02)
    Length: 10
  ⊟ Options: (6 bytes), Microsoft PPE/PPC
    ⊟ Microsoft PPE/PPC
        Type: Microsoft PPE/PPC (18)
        Length: 6
      ⊟ Supported Bits: 0x00000040, S
          .... ...0 .... .... .... .... .... .... = H: Stateless mode OFF
          .... .... .... .... .... .... 0... .... = M: 56-bit encryption OFF
          .... .... .... .... .... .... .1.. .... = S: 128-bit encryption ON
          .... .... .... .... .... .... ..0. .... = L: 40-bit encryption OFF
          .... .... .... .... .... .... ...0 .... = D: Obsolete (should ALWAYS be 0)
          .... .... .... .... .... .... .... ...0 = C: No desire to negotiate MPPC

0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 30 ac 3c 00 00 34 2f  88 e6 89 c2 04 f1 c0 a8   .0.<..4/ ........
0020  02 21 30 81 88 0b 00 0c  70 6d 00 00 00 0c 00 00   .!0..... pm......
0030  00 0a 80 fd 01 02 00 0a  12 06 00 00 00 40         ........ .....@
```

156 20.301177000 192.168.2.33 137.194.4.241 PPP CCP 62 Configuration Ack

MPPE: Microsoft Point to Point Encryption

Client

```
⊟ Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 137.194.4.241 (137.194.4.241)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 48
    Identification: 0x4b9b (19355)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x9d87 [validation disabled]
    Source: 192.168.2.33 (192.168.2.33)
    Destination: 137.194.4.241 (137.194.4.241)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3081
    Protocol Type: PPP (0x880b)
    Payload Length: 12
    Call ID: 34176
    Sequence Number: 13
    Acknowledgment Number: 12
⊟ Point-to-Point Protocol
    Protocol: Compression Control Protocol (0x80fd)
⊟ PPP Compression Control Protocol
    Code: Configuration Ack (2)
    Identifier: 2 (0x02)
    Length: 10
  ⊟ Options: (6 bytes), Microsoft PPE/PPC
    ⊟ Microsoft PPE/PPC
        Type: Microsoft PPE/PPC (18)
        Length: 6
      ⊟ Supported Bits: 0x00000040, S
          .... ...0 .... .... .... .... .... .... = H: Stateless mode OFF
          .... .... .... .... .... .... 0... .... = M: 56-bit encryption OFF
          .... .... .... .... .... .... .1.. .... = S: 128-bit encryption ON
          .... .... .... .... .... .... ..0. .... = L: 40-bit encryption OFF
          .... .... .... .... .... .... ...0 .... = D: Obsolete (should ALWAYS be 0)
          .... .... .... .... .... .... .... ...0 = C: No desire to negotiate MPPC

0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 30 4b 9b 00 00 80 2f  9d 87 c0 a8 02 21 89 c2   .0K..../ .....!..
0020  04 f1 30 81 88 0b 00 0c  85 80 00 00 00 0d 00 00   ..0..... ........
0030  00 0c 80 fd 02 02 00 0a  12 06 00 00 00 40         ........ .....@
```

RFC 3079 MPPE Key Derivation

2.5.3. Sample 128-bit Key Derivation Initial Values

Password = "clientPass"
Challenge = 10 2d b5 df 08 5d 30 41

Step 1: NtPasswordHash(Password, PasswordHash)
PasswordHash = 44 eb ba 8d 53 12 b8 d6 11 47 44 11 f5 69 89 ae

Step 2: PasswordHashHash = MD4(PasswordHash)
PasswordHashHash = 41 c0 0c 58 4b d2 d9 1c 40 17 a2 a1 2f a5 9f 3f

Step 3: GetStartKey(Challenge, PasswordHashHash, InitialSessionKey)
InitialSessionKey = a8 94 78 50 cf c0 ac ca d1 78 9f b6 2d dc dd b0

Step 4: Copy InitialSessionKey to CurrentSessionKey
CurrentSessionKey = a8 94 78 50 cf c0 ac c1 d1 78 9f b6 2d dc dd b0

Step 5: GetKey(InitialSessionKey, CurrentSessionKey, 16)
CurrentSessionKey = 59 d1 59 bc 09 f7 6f 1d a2 a8 6a 28 ff ec 0b 1e

IPCP:Internet Protocol Control Protocol

Client

```
154 20.300601000 192.168.2.33 137.194.4.241 PPP IPCP 74 Configuration Request
⊞ Frame 154: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊞ Ethernet II, Src: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54), Dst: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76)
⊟ Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 137.194.4.241 (137.194.4.241)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 60
    Identification: 0x4b9a (19354)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x9d7c [validation disabled]
    Source: 192.168.2.33 (192.168.2.33)
    Destination: 137.194.4.241 (137.194.4.241)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3081
    Protocol Type: PPP (0x880b)
    Payload Length: 24
    Call ID: 34176
    Sequence Number: 12
    Acknowledgment Number: 11
⊟ Point-to-Point Protocol
    Protocol: Internet Protocol Control Protocol (0x8021)
⊟ PPP IP Control Protocol
    Code: Configuration Request (1)
    Identifier: 8 (0x08)
    Length: 22
  ⊟ Options: (18 bytes), IP address, Primary DNS Server IP Address, Secondary DNS Server IP Address
    ⊟ IP address: 0.0.0.0
        Type: IP address (3)
        Length: 6
        IP Address: 0.0.0.0 (0.0.0.0)
    ⊞ Primary DNS Server IP Address: 0.0.0.0
    ⊞ Secondary DNS Server IP Address: 0.0.0.0

0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 3c 4b 9a 00 00 80 2f  9d 7c c0 a8 02 21 89 c2   .<K..../ .|...!..
0020  04 f1 30 81 88 0b 00 18  85 80 00 00 00 0c 00 00   ........ ........
0030  00 0b 80 21 01 08 00 16  03 06 00 00 00 00 81 06   ...!.... ........
0040  00 00 00 00 83 06 00 00  00 00                     ........ ..
```

IPCP:Internet Protocol Control Protocol

Serveur

```
160 20.342148000 137.194.4.241 192.168.2.33 PPP IPCP 74 Configuration Nak
⊞ Frame 160: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊞ Ethernet II, Src: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76), Dst: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54)
⊟ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 60
    Identification: 0xac3f (44095)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 52
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x88d7 [validation disabled]
    Source: 137.194.4.241 (137.194.4.241)
    Destination: 192.168.2.33 (192.168.2.33)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3081
    Protocol Type: PPP (0x880b)
    Payload Length: 24
    Call ID: 28781
    Sequence Number: 15
    Acknowledgment Number: 12
⊟ Point-to-Point Protocol
    Protocol: Internet Protocol Control Protocol (0x8021)
⊟ PPP IP Control Protocol
    Code: Configuration Nak (3)
    Identifier: 8 (0x08)
    Length: 22
  ⊟ Options: (18 bytes), IP address, Primary DNS Server IP Address, Secondary DNS Server IP Address
    ⊟ IP address: 137.194.20.176
        Type: IP address (3)
        Length: 6
        IP Address: 137.194.20.176 (137.194.20.176)
    ⊞ Primary DNS Server IP Address: 137.194.2.17
    ⊞ Secondary DNS Server IP Address: 255.255.255.255

0000   30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010   00 3c ac 3f 00 00 34 2f  88 d7 89 c2 04 f1 c0 a8   .<.?..4/ ........
0020   02 21 30 81 88 0b 00 18  70 6d 00 00 00 0f 00 00   .!0..... pm......
0030   00 0c 80 21 03 08 00 16  03 06 89 c2 14 b0 81 06   ...!.... ........
0040   89 c2 02 11 83 06 ff ff  ff ff                     ........ ..
```

134 20.203339000 137.194.4.241 192.168.2.33 PPP LCP 96 Identification

```
⊞ Frame 134: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface 0
⊞ Ethernet II, Src: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76), Dst: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54)
⊟ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 82
    Identification: 0xac34 (44084)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 52
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x88cc [validation disabled]
    Source: 137.194.4.241 (137.194.4.241)
    Destination: 192.168.2.33 (192.168.2.33)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3001
    Protocol Type: PPP (0x880b)
    Key: 0x0032706d
    Sequence Number: 4
⊟ Point-to-Point Protocol
    Address: 0xff
    Control: 0x03
    Protocol: Link Control Protocol (0xc021)
⊟ PPP Link Control Protocol
    Code: Identification (12)
    Identifier: 1 (0x01)
    Length: 46
    Magic Number: 0x8cd0d09d
    Message: user-ppp 3.4.2 (built COMPILATIONDATE)
```

Authentification

Identity Serveur

```
0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 52 ac 34 00 00 34 2f  88 cc 89 c2 04 f1 c0 a8   .R.4..4/ ........
0020  02 21 30 01 88 0b 00 32  70 6d 00 00 00 04 ff 03   .!0....2 pm......
0030  c0 21 0c 01 00 2e 8c d0  d0 9d 75 73 65 72 2d 70   .!...... ..user-p
0040  70 70 20 33 2e 34 2e 32  20 28 62 75 69 6c 74 20   pp 3.4.2  (built
0050  43 4f 4d 50 49 4c 41 54  49 4f 4e 44 41 54 45 29   COMPILAT IONDATE)
```

```
135 20.203341000 137.194.4.241 192.168.2.33 PPP CHAP 69 Challenge (NAME='', VALUE=0xc14785ae0cab8be3e6aab5b50bab9676)
⊞ Frame 135: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0
⊞ Ethernet II, Src: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76), Dst: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54)
⊟ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 55
    Identification: 0xac35 (44085)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 52
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x88e6 [validation disabled]
    Source: 137.194.4.241 (137.194.4.241)
    Destination: 192.168.2.33 (192.168.2.33)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3001
    Protocol Type: PPP (0x880b)
    Key: 0x0017706d
    Sequence Number: 5
⊟ Point-to-Point Protocol
    Protocol: Challenge Handshake Authentication Protocol (0xc223)
⊟ PPP Challenge Handshake Authentication Protocol
    Code: Challenge (1)
    Identifier: 1
    Length: 21
  ⊟ Data
      Value Size: 16
      Value: c147                                76

0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 37 ac 35 00 00 34 2f  88 e6 89 c2 04 f1 c0 a8   .7.5..4/ ........
0020  02 21 30 01 88 0b 00 17  70 6d 00 00 00 05 c2 23   .!0..... pm.....#
0030  01 01 00 15 10 c1 47                               ......G. ........
0040              76                                     ....v
```

Authentification

Serveur

Challenge

Authentification Client Response

```
140 20.206745000 192.168.2.33 137.194.4.241 PPP CHAP 107 Response (NAME='urien', VALUE=0xeb1825d756ac122b2f46d6668d2dc6fc0000000000000000...)
⊞ Frame 140: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
⊞ Ethernet II, Src: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54), Dst: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76)
⊟ Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 137.194.4.241 (137.194.4.241)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 93
    Identification: 0x4b92 (19346)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x9d63 [validation disabled]
    Source: 192.168.2.33 (192.168.2.33)
    Destination: 137.194.4.241 (137.194.4.241)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3001
    Protocol Type: PPP (0x880b)
    Key: 0x003d8580
    Sequence Number: 6
⊟ Point-to-Point Protocol
    Protocol: Challenge Handshake Authentication Protocol (0xc223)
⊟ PPP Challenge Handshake Authentication Protocol
    Code: Response (2)
    Identifier: 1
    Length: 59
  ⊟ Data
      Value Size: 49
      Value: eb1825d756ac122b2f46d6668d2dc6fc0000000000000000...
      Name: urien

0000  e4 f4 c6 3e a1 76 30 10  b3 4b 0f 54 08 00 45 00   ...>.v0. .K.T..E.
0010  00 5d 4b 92 00 00 80 2f  9d 63 c0 a8 02 21 89 c2   .]K..../ .c...!..
0020  04 f1 30 01 88 0b 00 3d  85 80 00 00 00 06 c2 23   ..0....= .......#
0030  02 01 00 3b 31 eb 18 25  d7 56 ac 12 2b 2f 46 d6   ...;1..% .V..+/F.
0040  66 8d 2d c6 fc 00 00 00  00 00 00 00 00 10 84 6d   f.-..... .......m
0050  48 7a 01 0c c7 99 8e 8b  ae 36 8e 46 48 0d 98 ba   Hz...... .6.FH...
0060  6d e8 60 0b 48 00 75 72  69 65 6e                  m.`.H.ur ien
```

142 20.254263000 137.194.4.241 192.168.2.33 PPP CHAP 98 Success (MESSAGE='S=B7750331D47749F15F8F72A8844BCE28A499527D')

```
⊞ Frame 142: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
⊞ Ethernet II, Src: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76), Dst: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54)
⊟ Internet Protocol Version 4, Src: 137.194.4.241 (137.194.4.241), Dst: 192.168.2.33 (192.168.2.33)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0xac36 (44086)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 52
    Protocol: Generic Routing Encapsulation (47)
  ⊞ Header checksum: 0x88c8 [validation disabled]
    Source: 137.194.4.241 (137.194.4.241)
    Destination: 192.168.2.33 (192.168.2.33)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊞ Flags and Version: 0x3081
    Protocol Type: PPP (0x880b)
    Payload Length: 48
    Call ID: 28781
    Sequence Number: 6
    Acknowledgment Number: 6
⊟ Point-to-Point Protocol
    Protocol: Challenge Handshake Authentication Protocol (0xc223)
⊟ PPP Challenge Handshake Authentication Protocol
    Code: Success (3)
    Identifier: 1
    Length: 46
    Message: S=B7750331D47749F15F8F72A8844BCE28A499527D
```

Authentification Serveur Success

```
0000  30 10 b3 4b 0f 54 e4 f4  c6 3e a1 76 08 00 45 00   0..K.T.. .>.v..E.
0010  00 54 ac 36 00 00 34 2f  88 c8 89 c2 04 f1 c0 a8   .T.6..4/ ........
0020  02 21 30 81 88 0b 00 30  70 6d 00 00 00 06 00 00   .!0....0 pm......
0030  00 06 c2 23 03 01 00 2e  53 3d 42 37 37 35 30 33   ...#.... S=B77503
0040  33 31 44 34 37 37 34 39  46 31 35 46 38 46 37 32   31D47749 F15F8F72
0050  41 38 38 34 34 42 43 45  32 38 41 34 39 39 35 32   A8844BCE 28A49952
0060  37 44                                              7D
```

Transport GRE Chiffré

```
⊞ Ethernet II, Src: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54), Dst: Netgear_3e:a1:76 (e4:f4:c6:3e:a1:76)
⊟ Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 137.194.4.241 (137.194.4.241)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 92
    Identification: 0x4ba2 (19362)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: Generic Routing Encapsulation (47)
  ⊟ Header checksum: 0x9d54 [validation disabled]
      [Good: False]
      [Bad: False]
    Source: 192.168.2.33 (192.168.2.33)
    Destination: 137.194.4.241 (137.194.4.241)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Generic Routing Encapsulation (PPP)
  ⊟ Flags and Version: 0x3001
      0... .... .... .... = Checksum Bit: No
      .0.. .... .... .... = Routing Bit: No
      ..1. .... .... .... = Key Bit: Yes
      ...1 .... .... .... = Sequence Number Bit: Yes
      .... 0... .... .... = Strict Source Route Bit: No
      .... .000 .... .... = Recursion control: 0
      .... .... 0000 0... = Flags (Reserved): 0
      .... .... .... .001 = Version: Enhanced GRE (1)
    Protocol Type: PPP (0x880b)
    Key: 0x003c8580
    Sequence Number: 17
⊟ Point-to-Point Protocol

0010  00 5c 4b a2 00 00 80 2f  9d 54 c0 a8 02 21 89 c2   .\K..../ .T...!..
0020  04 f1 30 01 88 0b 00 3c  85 80 00 00 00 11 fd 90   ..0....< ........
0030  01 68 7c 18 07 f0 51 3b  02 71 fb 43 6a d9 7b 6e   .h|...Q; .q.Cj.{n
0040  83 50 20 62 f4 67 b2 77  73 c3 33 e3 55 17 21 f7   .P b.g.w s.3.U.!.
0050  ed 2e ef 83 a0 a7 a3 d6  13 17 37 25 83 c3 05 cc   ........ ..7%....
0060  f1 5f c9 b0 c4 cb a4 d9  08 60                     ._...... .`
```

Chiffrement RC4 sans contrôle d'intégrité !

Au sujet de EAP
Extensible Authentication Protocol

EAP, what else ?

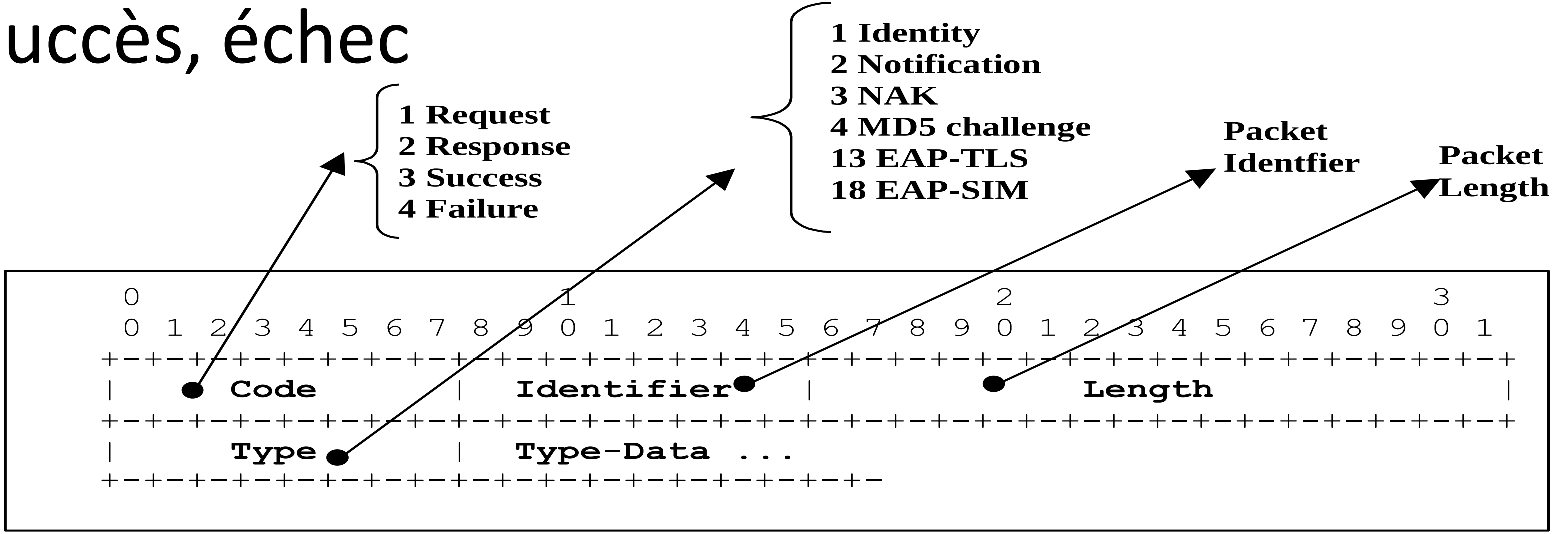
- The Extensible Authentication Protocol (EAP) was introduced in 1999, in order to define a flexible authentication framework.
- EAP, RFC 3748, "Extensible Authentication Protocol, (EAP)", June 2004.
  - EAP-TLS, RFC 2716, "PPP EAP TLS Authentication Protocol“, 1999.
  - EAP-SIM, RFC 4186, " Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) ", 2006
  - EAP-AKA, RFC 4187, " Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA) “, 2006

EAP Applications

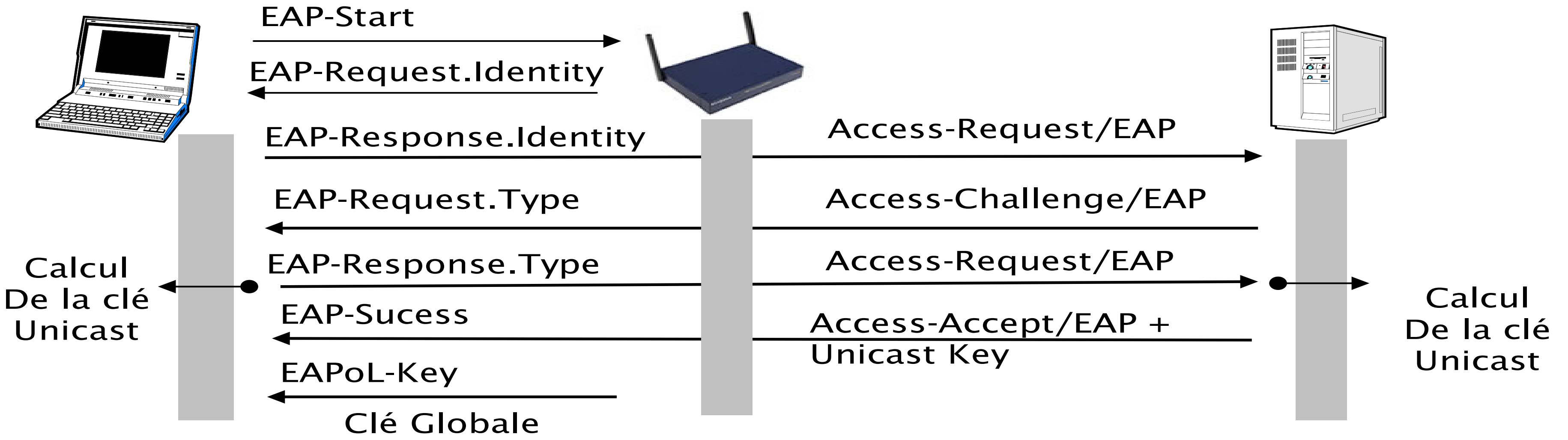
- Wireless LAN
  - Wi-Fi, IEEE 802.1x, 2001
  - WiMAX mobile, IEEE 802.16e , PKM-EAP, 2006
- Wired LANs
  - ETHERNET, IEEE 802.3
  - PPP, RFC 1661, “The Point-to-Point Protocol (PPP)”, 1994
- VPN (Virtual Private Network) technologies
  - PPTP, Point-to-Point Tunneling Protocol (PPTP), RFC 2637
  - L2TP, Layer Two Tunneling Protocol (L2TP), RFC 2661
  - IKEv2, RFC 4306, "Internet Key Exchange (IKEv2) Protocol“, 2005
- Authentication Server
  - RADIUS, RFC 3559, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)”, 2003
  - DIAMETER, RFC 4072, "Diameter Extensible Authentication Protocol Application“, 2005
- Voice Over IP
  - UMA, Unlicensed Mobile Access, http://www.umatechnology.org

Le protocole EAP.

- EAP est conçu pour transporter des scénarios d authentification.
- Quatre types de messages, requêtes, réponses, succès, échec

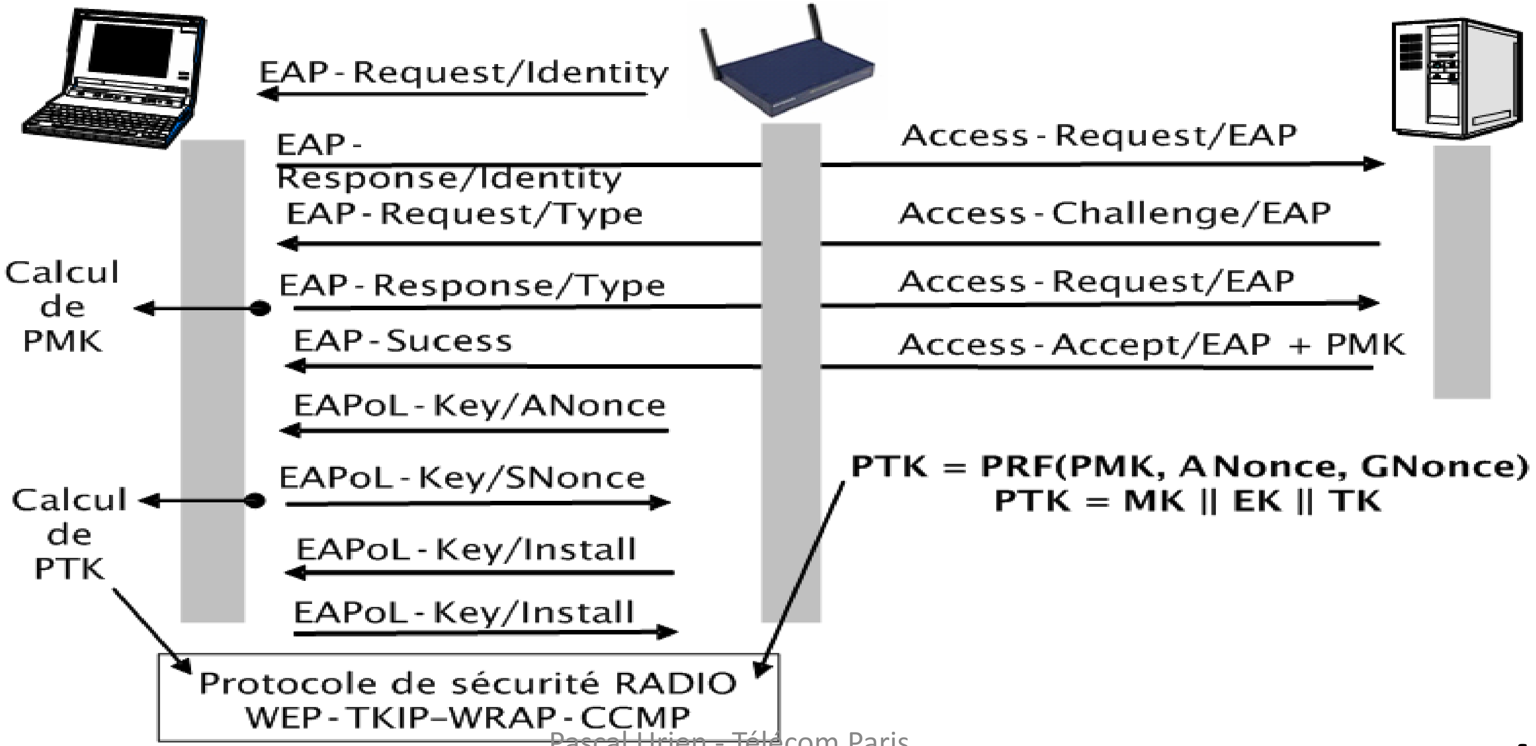


IEEE 802.1x


EAP-Start
EAP-Request.Identity
EAP-Response.Identity
Access-Request/EAP
EAP-Request.Type
Access-Challenge/EAP
Calcul
De la clé
Unicast
EAP-Response.Type
Access-Request/EAP
Calcul
De la clé
Unicast
EAP-Sucess
Access-Accept/EAP +
Unicast Key
EAPoL-Key
Clé Globale

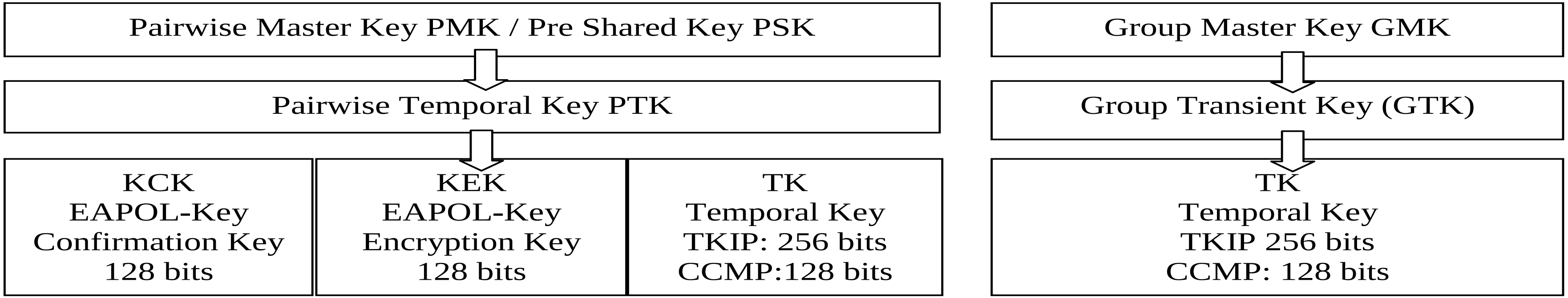
IEEE 802.11i : Distribution des clés

- Four ways handshake (PTK).
- Two ways hanshake (GTK).



802.11 i: Hiérarchie des clés

- PMK est déduite de l’authentification EAP.
- PSK est une alternative à PMK.
- GMK est une clé maître de groupe.



SSH

SSH

- La première version de SSH (SSH-1) a été conçue par Tatu Ylönen, à Espoo, en Finlande en 1995.
- La version suivante a été nommée SSH-2. Le groupe de recherche de l'IETF « secsh » a défini en janvier 2006 le standard Internet SSH-2
  - RFC 4251, Secure Shell (SSH) Protocol Architecture
  - RFC 4253, The Secure Shell (SSH) Transport Layer Protocol
  - RFC 4252, The Secure Shell (SSH) Authentication Protocol
  - RFC 4254, The Secure Shell (SSH) Connection Protocol

RFC 4251: Protocol Architecture

- Transport Layer Protocol
  - Réalise l’authentification du serveur, la confidentialité et l’intégrité des messages SSH
- User Authentication Protocol
  - Réalise l’authentification du client
  - Les messages sont acheminés via le canal sécurisé mis en place par le Transport Layer Protocol
- Connection Protocol
  - Multiplexe plusieurs canaux logiques dans le tunnel sécurisé.
  - Les messages sont transportés par le User Authentication Protocol

RFC 4251: Type de données

- byte: un octet
- boolean: un octet
- uint32: 4 octets (unsigned integer)
- uint64: 8 octets (unsigned integer)
- string: uint32 (longueur) [caractères ASCII]
- mpint: signed integer, encodé comme un string
- name-list: encodé comme un string, comportant une liste de noms séparés par une virgule (0x2c)

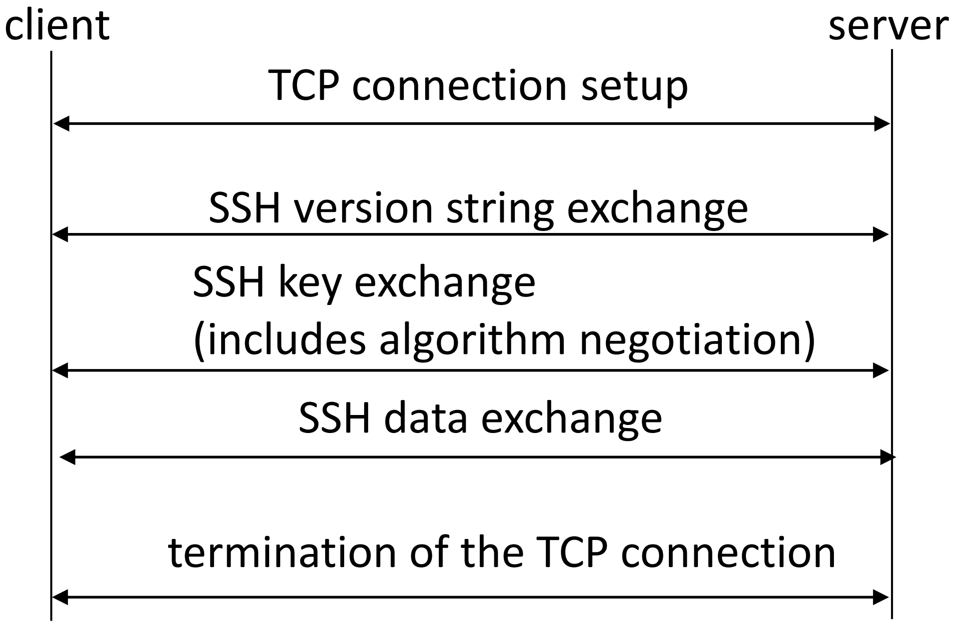
Structure d'un paquet TLP (Transport Layer Protocol)

CHIFFREMENT

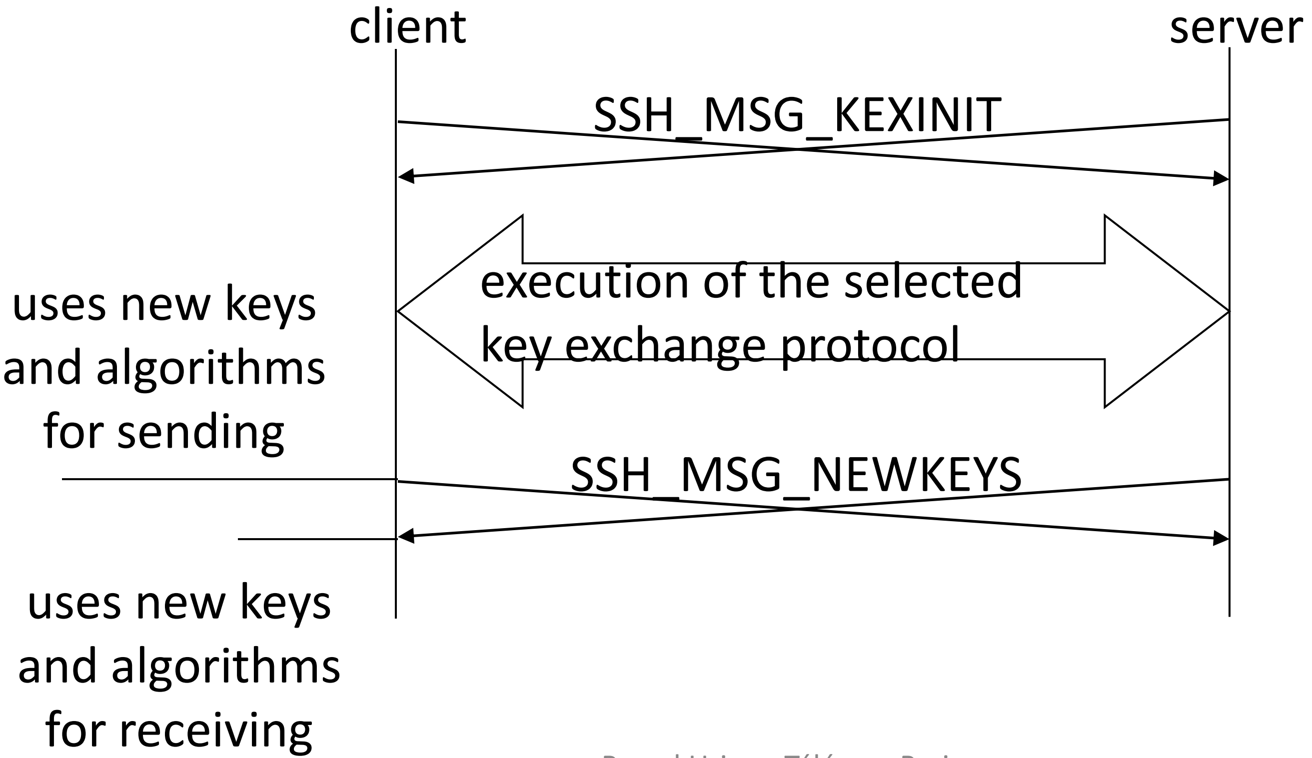
- uint32 packet_length
  - sans le champ longueur et le MAC
- byte padding_length
- byte[n1] payload;
  - n1 = packet_length - padding_length – 1
  - Le premier octet du payload est le MESSAGE NUMBER
- byte[n2] random padding;
  - n2 = padding_length
- byte[m] mac (Message Authentication Code - MAC)
  - m = mac_length
  - MAC(key, sequence_number || unencrypted_packet)

HMAC

SSH TLP(Transport Layer Protocol) RFC 4253



Key Exchange - RFC 4253



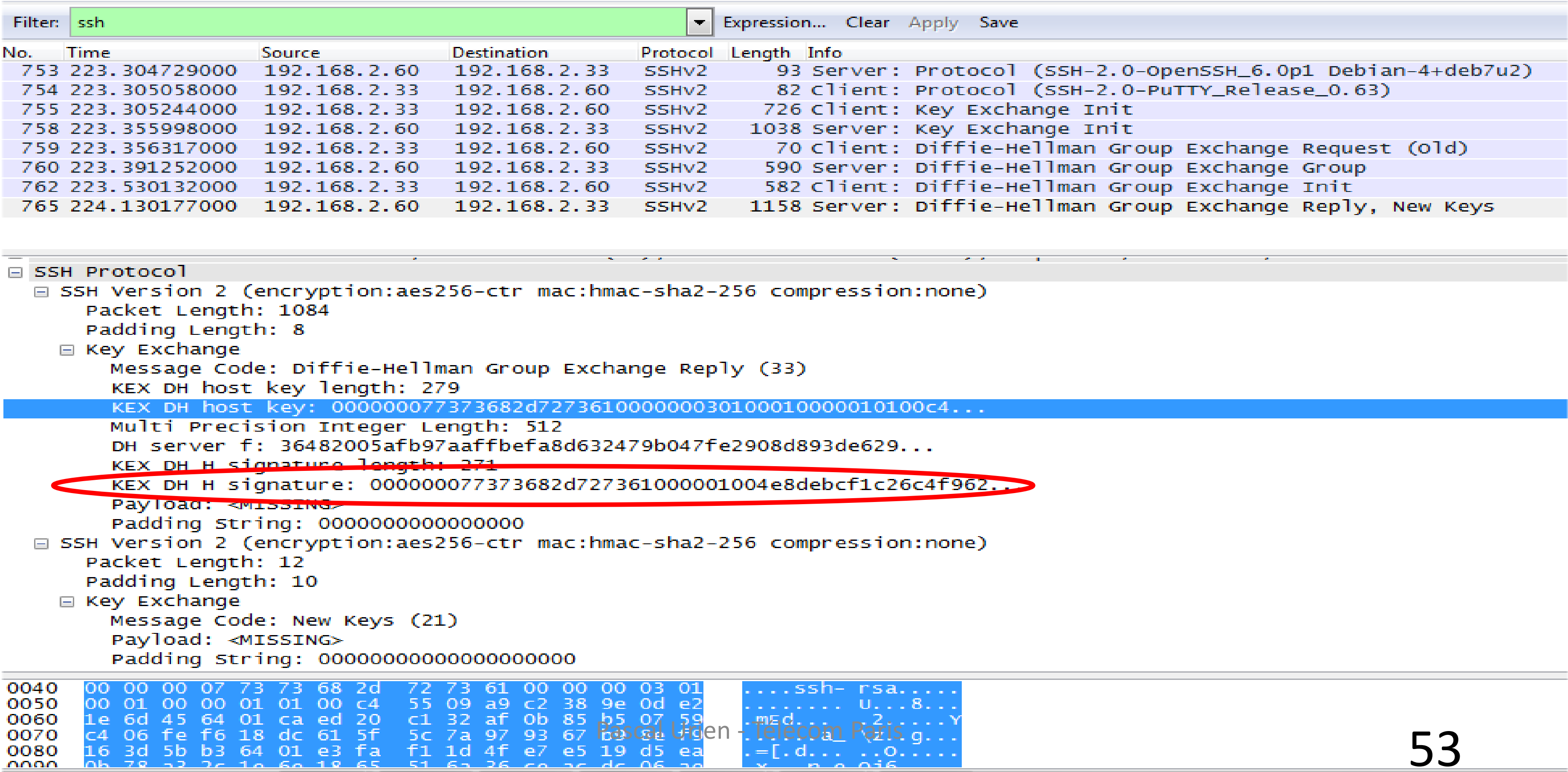
Signature du Serveur

- C (client) generates a random number x (1 < x < p) and computes
  - e = g^x mod p. C sends e to S.
- S (server)generates a random number y (0 < y < p) and computes
  - f = g^y mod p. S receives e. It computes K = e^y mod p,
  - H = hash(V_C || V_S || I_C || I_S || K_S || **e || f || K**)
  - And a **signature** on H with its private host key

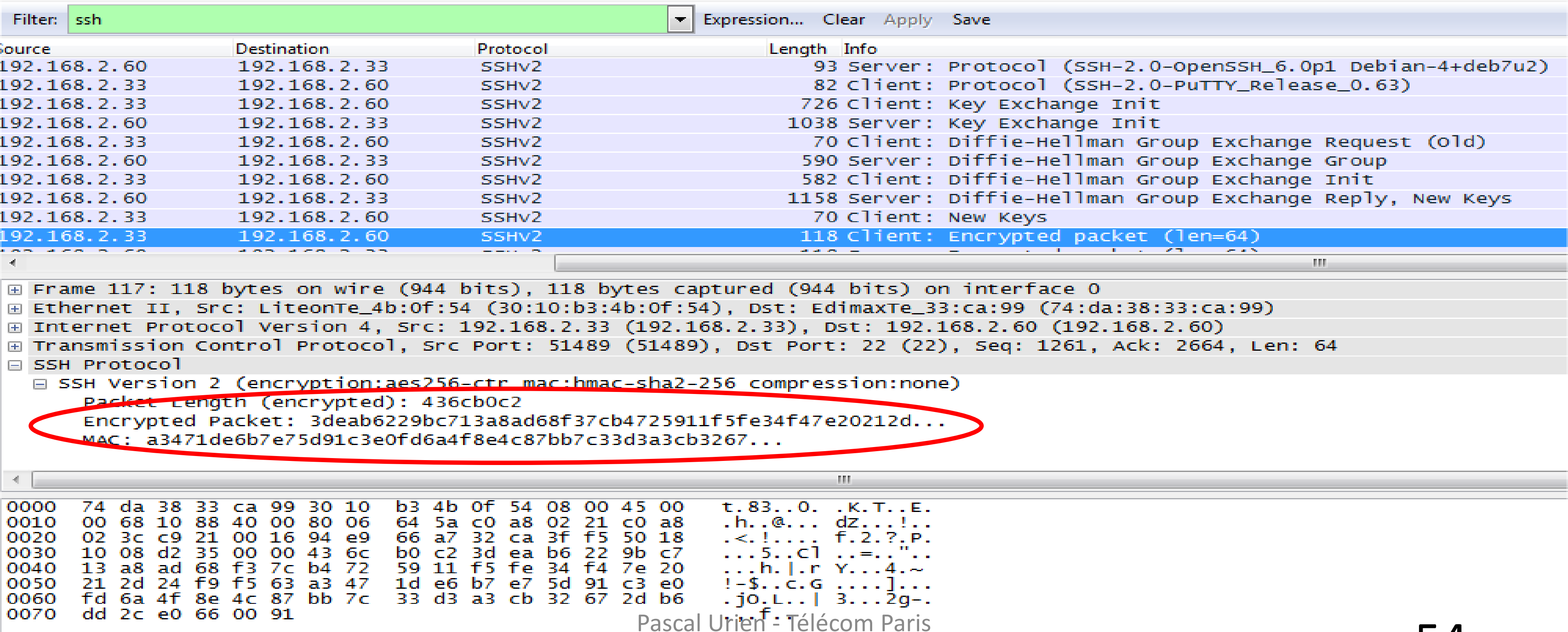
Rappel: Signature DSA

- La clé DSA publique comporte 4 paramètres   y ,p, q, g
  - q un premier premier de N bits
  - p un nombre premier de L bits tel que p-1 soit un multiple de q, p= 1 + aq
    - Exemple de couples (L,N): (1024, 160), (2048, 224), (2048, 256), (3072, 256)
  - $g = h^{(p-1)/q}$ mod p, on choisit h=2 en règle générale
  - Clé publique $y = g^x$ mod p
    - clé privée x tel que; 0<x<q
- La signature DSA est le couple (r,s)
  - On choisit un nombre aléatoire k
    - tel que 0<k<q
  - On calcule = (g^k mod p) mod q, r doit être non nul
  - s = k^{-1} (H(m)+ x.r) mod q, s doit être non nul
- DSA-Sig-Value ::= SEQUENCE { r INTEGER, s INTEGER }

Signature du Serveur

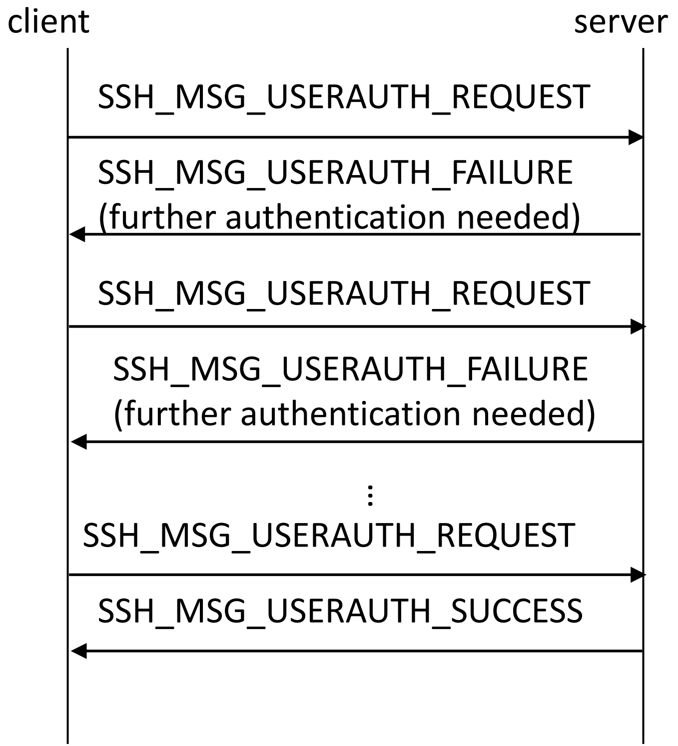


Tunnel SSH



Filter: ssh
Expression... Clear Apply Save
Source Destination Protocol Length Info
192.168.2.60 192.168.2.33 SSHv2 93 Server: Protocol (SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2)
192.168.2.33 192.168.2.60 SSHv2 82 Client: Protocol (SSH-2.0-PuTTY_Release_0.63)
192.168.2.33 192.168.2.60 SSHv2 726 Client: Key Exchange Init
192.168.2.60 192.168.2.33 SSHv2 1038 Server: Key Exchange Init
192.168.2.33 192.168.2.60 SSHv2 70 Client: Diffie-Hellman Group Exchange Request (old)
192.168.2.60 192.168.2.33 SSHv2 590 Server: Diffie-Hellman Group Exchange Group
192.168.2.33 192.168.2.60 SSHv2 582 Client: Diffie-Hellman Group Exchange Init
192.168.2.60 192.168.2.33 SSHv2 1158 Server: Diffie-Hellman Group Exchange Reply, New Keys
192.168.2.33 192.168.2.60 SSHv2 70 Client: New Keys
192.168.2.33 192.168.2.60 SSHv2 118 Client: Encrypted packet (len=64)
Frame 117: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
Ethernet II, Src: LiteonTe_4b:0f:54 (30:10:b3:4b:0f:54), Dst: EdimaxTe_33:ca:99 (74:da:38:33:ca:99)
Internet Protocol Version 4, Src: 192.168.2.33 (192.168.2.33), Dst: 192.168.2.60 (192.168.2.60)
Transmission Control Protocol, Src Port: 51489 (51489), Dst Port: 22 (22), Seq: 1261, Ack: 2664, Len: 64
SSH Protocol
SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
Packet Length (encrypted): 436cb0c2
Encrypted Packet: 3deab6229bc713a8ad68f37cb4725911f5fe34f47e20212d...
MAC: a3471de6b7e75d91c3e0fd6a4f8e4c87bb7c33d3a3cb3267...
0000 74 da 38 33 ca 99 30 10 b3 4b 0f 54 08 00 45 00 t.83..0. .K.T..E.
0010 00 68 10 88 40 00 80 06 64 5a c0 a8 02 21 c0 a8 .h..@... dZ...!..
0020 02 3c c9 21 00 16 94 e9 66 a7 32 ca 3f f5 50 18 .<.!.... f.2.?.P.
0030 10 08 d2 35 00 00 43 6c b0 c2 3d ea b6 22 9b c7 ...5..Cl ..=.."..
0040 13 a8 ad 68 f3 7c b4 72 59 11 f5 fe 34 f4 7e 20 ...h.|.r Y...4.~
0050 21 2d 24 f9 f5 63 a3 47 1d e6 b7 e7 5d 91 c3 e0 !-$..c.G ....]...
0060 fd 6a 4f 8e 4c 87 bb 7c 33 d3 a3 cb 32 67 2d b6 .jO.L..| 3...2g-.
0070 dd 2c e0 66 00 91

RFC 4252 User Authentication Protocol



- Trois méthodes
  - Password
    - Echange du mot de passe
  - PublicKey
    - Signature du client basée sur un mécanisme asymétrique (DSS) à l’aide d’une clé privée du client (certificat client)
  - Host Based
    - Signature basée sur un mécanisme asymétrique (DSS) à l’aide d’une clé privée du host (certificat host)

RFC 4254 Connection Protocol

- Gestion de canaux logiques identifiés par un type
  - OPEN, CLOSE
  - REQUEST, RESPONSE
  - Gestion de (sous) type (shell, X11, …) dans un canal
- Ouverture de sessions interactives, exécution distante d’un programme
  - Shell, X11
- TCP/IP Port Forwarding
  - Redirection de canaux logiques SSH vers des ports TCP/IP.

IPSEC

IPSEC: AH et ESP

- Deux en têtes spécifiques sont utilisés, AH (IP Authentification Header) et ESP (IP Encapsulating Security Payload).
- AH garantit l'intégrité et l'authentification des datagrammes IP, mais n'assure pas la confidentialité des données.
- ESP est utilisé pour fournir l'intégrité, l'authentification et la confidentialité des datagrammes IP.

Security Association

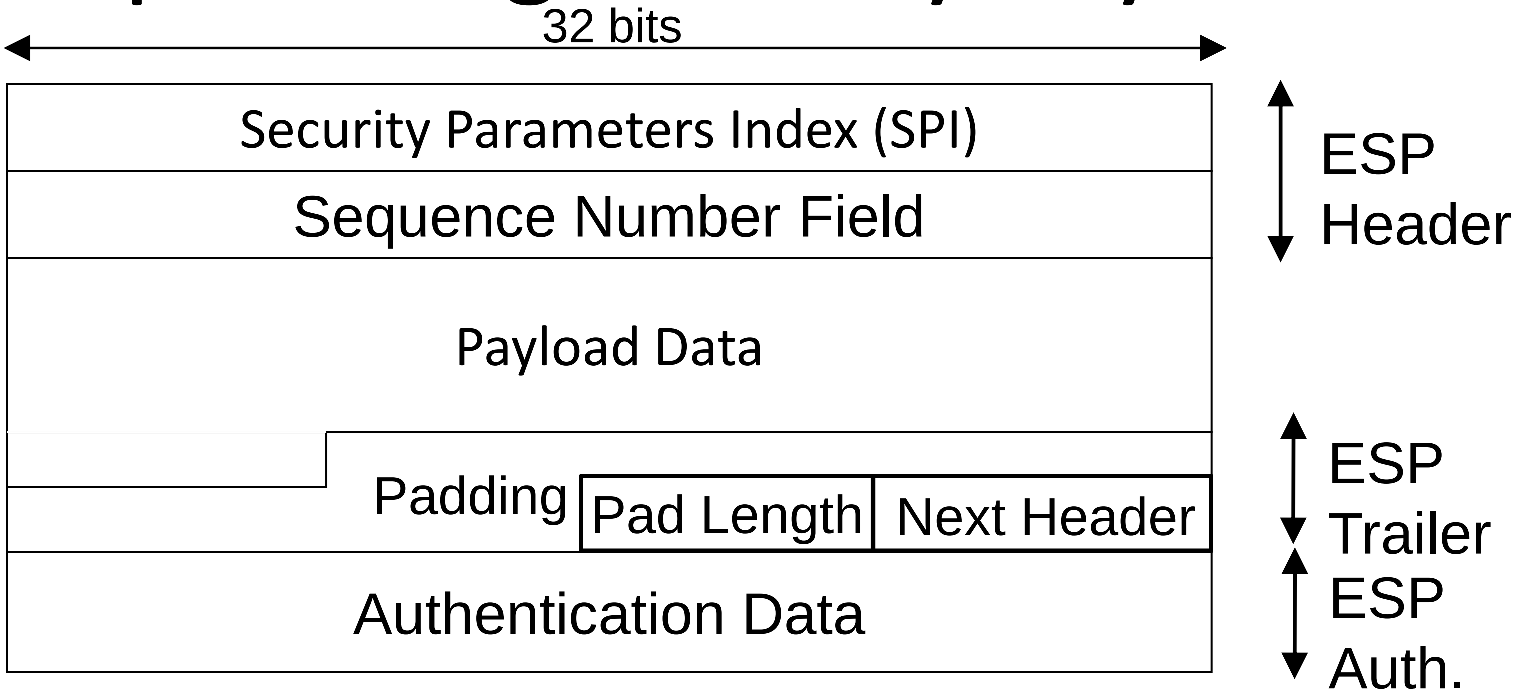
- Ce concept est fondamental à la fois pour AH et ESP. La combinaison d'un SPI (Security Parameter Index) et d'une adresse de destination identifie de manière unique un SA particulier.
- Une association de sécurité inclue usuellement les paramètres suivant :
  - Un algorithme d'authentification (utilisé pour AH).
  - La (les) clé(s) utilisée(s) par l'algorithme d'authentification.
  - L'algorithme de chiffrement utilisé par ESP.
  - La (les) clé(s) utilisée(s) par l'algorithme de chiffrement.
  - Divers paramètres utiles à l'algorithme de chiffrement.
  - L'algorithme d'authentification utilisé avec ESP (s'il existe)
  - Les clés utilisées avec l'algorithme d'authentification d'ESP (si nécessaire).
  - La durée de vie de la clé.
  - La durée de vie du SA.
  - La ou les adresses de source du SA
  - Le niveau de sécurité (Secret, non classé ...)
- Le système hôte qui émet l'information sélectionne un SA en fonction du destinataire. L'association de sécurité est de manière générale mono directionnelle.

Authentication Header

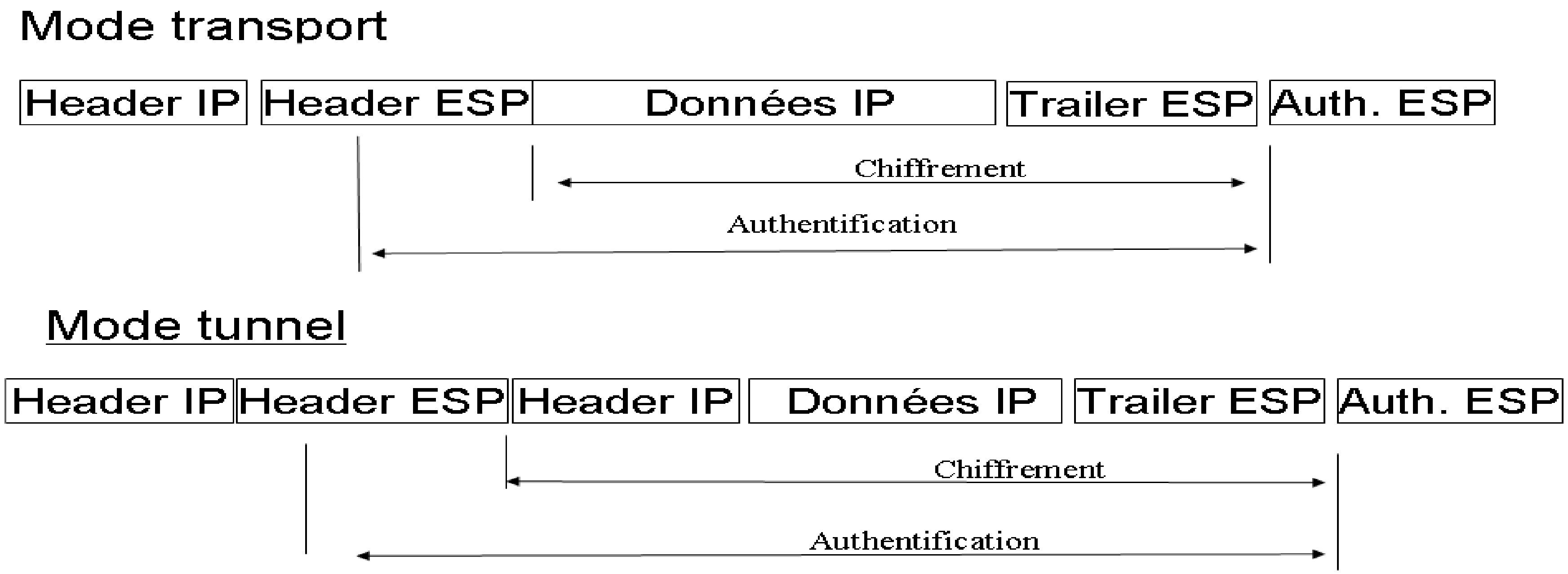
32 bits

Next Header	Payload Length	RESERVED
Security Parameters Index (SPI)		
Sequence Number Field		
Authentication Data (longueur variable)		

Encapsulating Security Payload



IPSEC: Mode Transport et Mode Tunnel



Au sujet de IKEv1

- Internet Key Exchange
- RFC 2409, 1998
- IKE PHASE 1 réalise une association de sécurité ISAKMP entre deux systèmes, qui protège les échanges de IKE phase 2
  - 4 modes, Main Mode, Agressive Mode, Quick Mode, New Group Mode
  - Plusieurs protocoles d'échanges de clés
    - Asymétriques, OAKLEY et SKEME
    - Symétrique (Pre-Shared-Key)
- IKE PHASE 2 réalise une association de sécurité pour des sessions IPSEC

IKEv1, Pre-Shared-Keys, Main Mode

```
        Initiator                               Responder
       -----------                             -----------       Echange
        HDR, SA                     -->                          Chiffré
                                    <--    HDR, SA
        HDR, KE, Ni                 -->
                                    <--    HDR, KE, Nr
        HDR*, IDii, [ CERT, ] SIG_I -->
                                    <--    HDR*, IDir, [ CERT, ] SIG_R
```

```
For pre-shared keys:
              SKEYID = prf(pre-shared-key, Ni_b | Nr_b)

The result of either Main Mode or Aggressive Mode is three groups of
authenticated keying material:

      SKEYID_d = prf(SKEYID, g^xy | CKY-I | CKY-R | 0)
      SKEYID_a = prf(SKEYID, SKEYID_d | g^xy | CKY-I | CKY-R | 1)
      SKEYID_e = prf(SKEYID, SKEYID_a | g^xy | CKY-I | CKY-R | 2)

   and agreed upon policy to protect further communications. The values
   of 0, 1, and 2 above are represented by a single octet. The key used
   for encryption is derived from SKEYID_e in an algorithm-specific
   manner.

   To authenticate either exchange the initiator of the protocol
   generates HASH_I(SIG_I) and the responder generates HASH_R(SIG_R)

where:

    HASH_I = prf(SKEYID, g^xi | g^xr | CKY-I | CKY-R | SAi_b | IDii_b )
    HASH_R = prf(SKEYID, g^xr | g^xi | CKY-R | CKY-I | SAi_b | IDir_b )
```

SAi_b is the entire body of the SA payload (minus the ISAKMP generic header), all proposals and all transforms offered by the Initiator.
CKY-I and CKY-R are the ... ookie, respectively, from the ISAKMP header.
g^xi and g^xr are the Di ... tor and responder respectively.

IKeV1, Phase II, Pre-Shared-Key, Quick Mode

```
Initiator                                                                   Responder
-----------                                                                -----------
HDR*, HASH(1), SA, Ni [, KE ] [, IDci, IDcr ] -->
                                        <-- HDR*, HASH(2), SA, Nr [, KE ] [, IDci, IDcr]
HDR*, HASH(3) -->
```

```
HASH(1) = prf(SKEYID_a, M-ID | SA | Ni [ | KE ] [ | IDci | IDcr ] )
HASH(2) = prf(SKEYID_a, M-ID | Ni_b | SA | Nr [ | KE ] [ | IDci | IDcr ] )
HASH(3) = prf(SKEYID_a, 0 | M-ID | Ni_b | Nr_b)
```

```
KEYMAT = prf(SKEYID_d, protocol | SPI | Ni_b | Nr_b)
```

IDci, IDcr, identités, les adresses IP en fait.
M-ID, identifiant du message, extrait de l’en tête ISAKMP

SSL/TLS

Historique

- SSL défini par *netscape* et intégré au browser
  - Première version de SSL testé en interne
  - Première version de SSL diffusé : V2 (1994)
  - Version actuelle V3
- Standard à l ’IETF au sein du groupe Transport Layer Security (TLS)
- The TLS Protocol Version 1.0", RFC 2246, January 1999
- "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006
- " Datagram Transport Layer Security", RFC 4347, April 2006
- "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008
- "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012

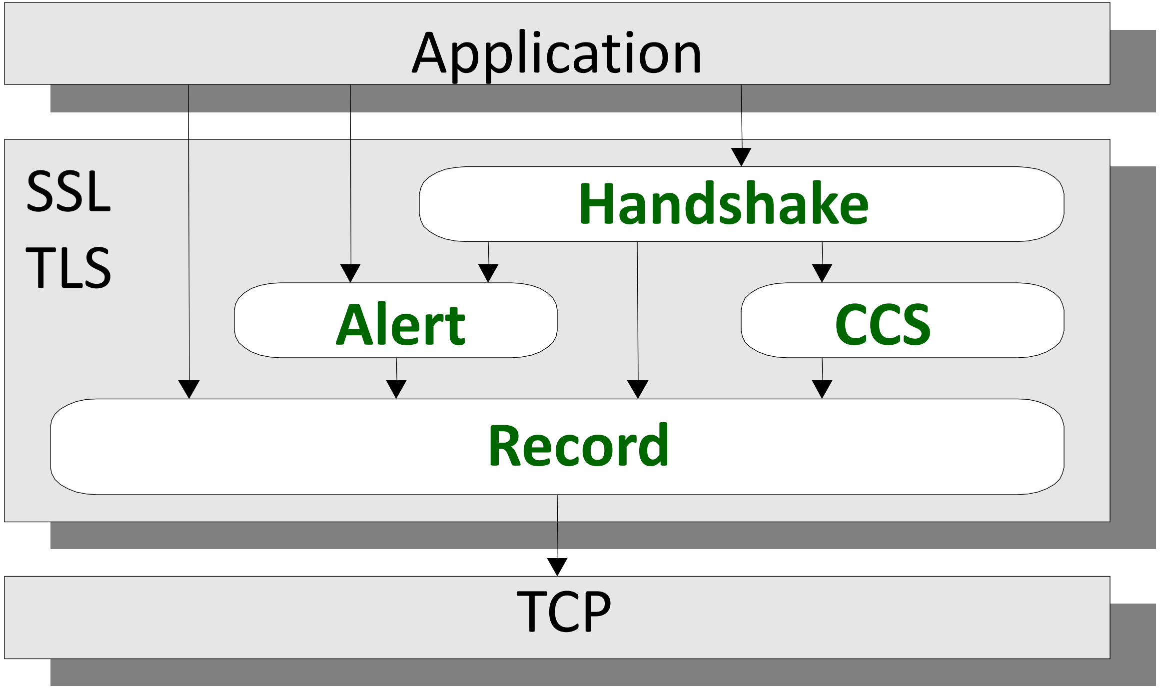
SSL : Services

- Authentification
  - Serveur (obligatoire), client (optionnel)
  - Utilisation de certificat X509 V3
  - A l'établissement de la session.
- Confidentialité
  - Algorithme de chiffrement symétrique négocié, clé généré à l'établissement de la session.
- Intégrité
  - Fonction de hachage avec clé secrète : HMAC(clé secrète,Message)
- Non Rejeu
  - Numéro de séquence

Version de TLS

- TLS 1.0
  - Structure des messages est compatible avec SSLv2, SSLv 3
  - Les fonctions cryptographiques (PRF…) sont différentes de celles de SSL
  - MD5, SHA1, HMAC-MD5, HMAC-SHA1
  - DH, RSA
  - Mode CBC avec IV fixe
- TLS 1.1
  - Les extensions TLS sont supportées (rfc 4366, rfc 6066)
  - En mode CBC le IV est généré pour chaque Record Packet et transmis
- TLS 1.2
  - Support de hmac_sha256, hmac_sha384, hmac_sha512 pour le PRF et le HMAC
  - Signature sha256, sha384, sha512 pour le client
  - Les courbes elliptiques sont supportées

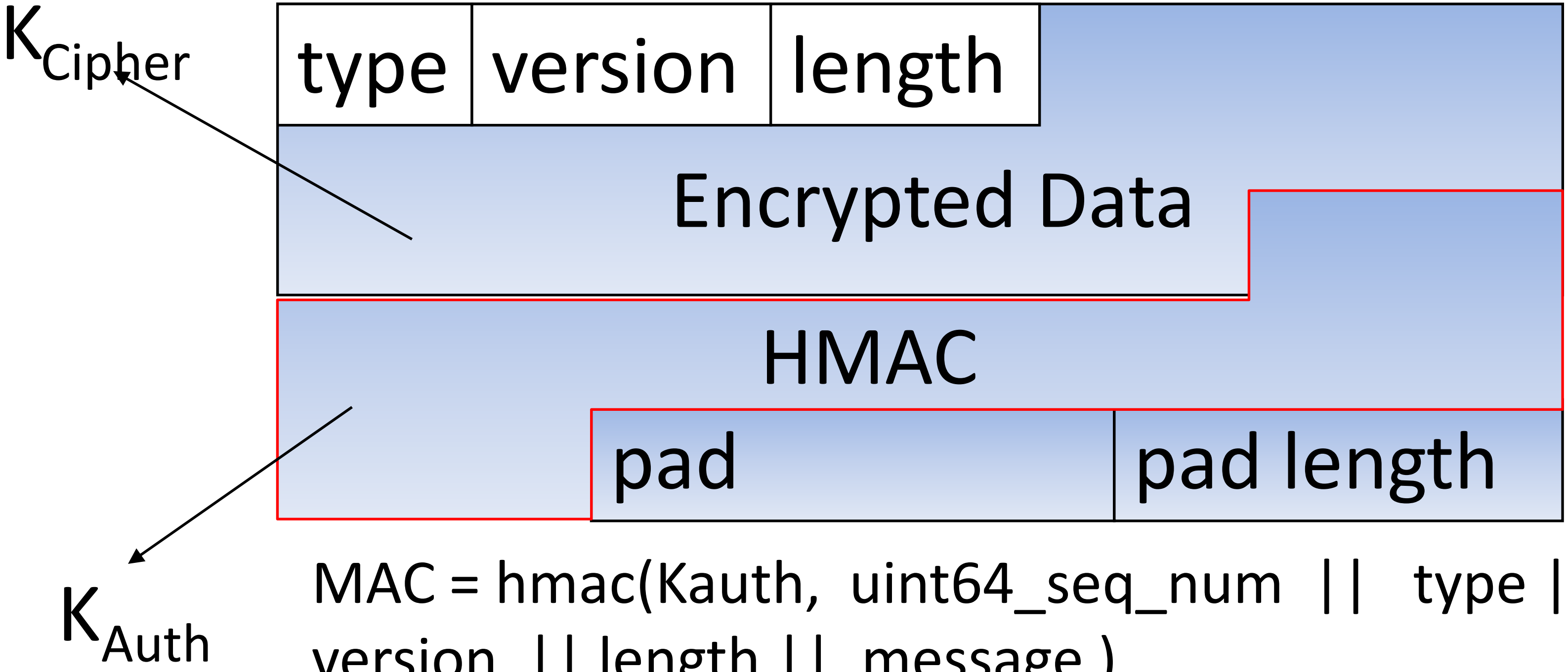
SSL/TLS : Protocoles



Record Layer

Le HMAC et les padding bytes sont chiffrés

Le calcul HMAC ne comprend pas les octets de padding



MAC = hmac(Kauth, uint64_seq_num || type || version || length || message)

Flights TLS : Full Mode

```
 Client                                    Serveur
 ------                                    -------
 flight1 ->
(TLS client-hello)
                                       <- flight 2
                                       (TLS server-hello,
                                        TLS certificate,
                                        TLS server-key-exchange,
                                        TLS certificate-request,
                                        TLS server-hello-done)
flight 3 ->
(TLS certificate,
 TLS client-key-exchange,
 TLS certificate-verify,
 TLS change-cipher-spec,
 TLS finished)
                                        <- flight 4
                                       (TLS change-cipher-spec,
                                        TLS finished)
```

Flights TLS: Resume Mode

```
   Client                                   Serveur
   ------                                   -------
  flight1 ->
 (TLS client-hello)

                                        <- flight 2
                                        (TLS server-hello,
                                         TLS change-cipher-spec,
                                         TLS finished)

 flight 3 ->
 (TLS change-cipher-spec,
  TLS finished)
```

Key Exchange

```
struct {

select (KeyExchangeAlgorithm)
{
  case diffie_hellman:
  ServerDHParams params;
  Signature signed_params;

  case rsa:
  ServerRSAParams params;
  Signature signed_params; };

      } ServerKeyExchange;
```

```
struct {

select (KeyExchangeAlgorithm)
{
 case rsa: EncryptedPreMasterSecret;

 case diffie_hellman: DiffieHellmanClientPublicValue;

} exchange_keys;

        } ClientKeyExchange;
```

The ServerKeyExchange message is sent by the server only when the server Certificate message (if sent) does not contain enough data to allow the client to exchange a premaster secret. This is true for the following key exchange methods: DHE_DSS DHE_RSA, DH_anon

RFC 5246, TLS1.2

- TLS supports three authentication modes:
  - authentication of both parties,
  - Server authentication with an unauthenticated client,
  - and total anonymity.

RFC 5246, TLS1.2

- Anonymous Key Exchange
  - Completely anonymous sessions can be established using Diffie-Hellman for key exchange.
  - The server's public parameters are contained in the server key exchange message, and the client's are sent in the client key exchange message
- RSA Key Exchange and Authentication
  - With RSA, key exchange and server authentication are combined. The public key is contained in the server's certificate.
  - When RSA is used for key exchange, clients are authenticated using the certificate verify message.
- Diffie-Hellman Key Exchange with Authentication
  - When Diffie-Hellman key exchange is used, the server can either supply a certificate containing fixed Diffie-Hellman parameters or use the server key exchange message to send a set of temporary Diffie-Hellman parameters signed with a DSA or RSA certificate
  - If the client has a certificate containing fixed Diffie-Hellman parameters, its certificate contains the information required to complete the key exchange.

RFC 5246, TLS1.2

- RSA-Encrypted Premaster Secret Message
  - struct { ProtocolVersion client_version; opaque random[46]; } PreMasterSecret;
- Diffie-Hellman
  - A conventional Diffie-Hellman computation is performed. The negotiated key (Z) is used as the pre_master_secret, and is converted into the master_secret. Leading bytes of Z that contain all zero bits are stripped before it is used as the pre_master_secret.

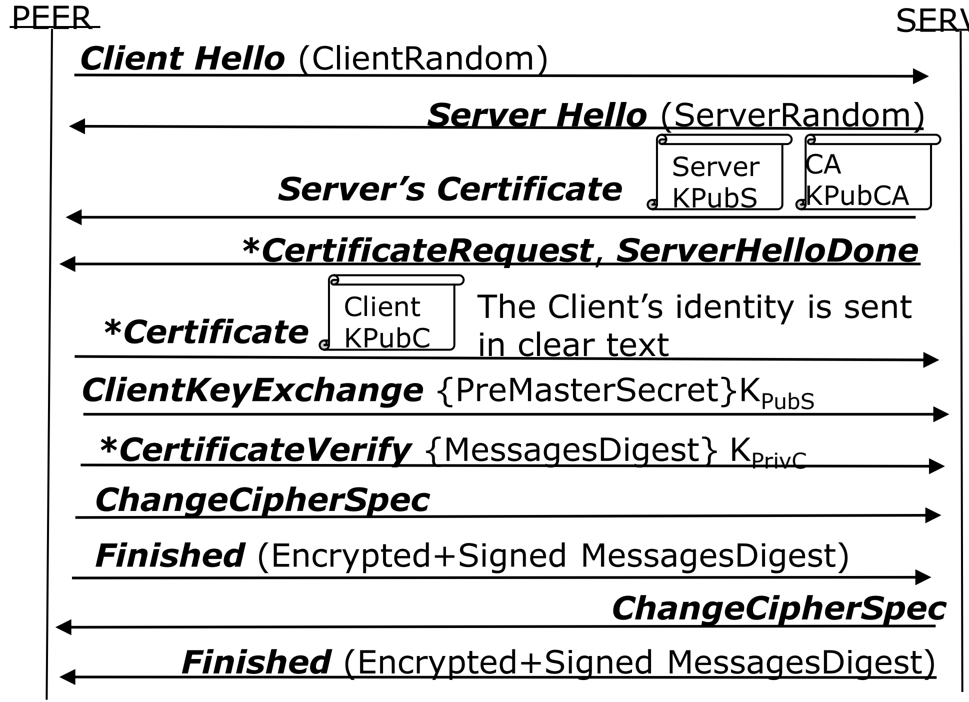
RFC4279: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)

- The premaster secret is formed as follows: if the PSK is N octets long, concatenate a uint16 with the value N, N zero octets, a second uint16 with the value N, and the PSK itself.
- All the ciphersuites in this document share the same general structure for the premaster secret, namely,
  - struct { opaque other_secret<0..2^16-1>;opaque psk<0..2^16-1>; };
- Here "other_secret" either is zeroes (plain PSK case) or comes from the Diffie-Hellman or RSA exchange (DHE_PSK and RSA_PSK, respectively).

Messages Handshake

```
struct
{ HandshakeType msg_type;
 /* handshake type */ uint24 length;
 /* bytes in message */
 select (HandshakeType)
{ case hello_request: HelloRequest;
   case client_hello:    ClientHello;
   case server_hello:   ServerHello;
   case certificate:  Certificate;
   case server_key_exchange:  ServerKeyExchange;
   case certificate_request: CertificateRequest;
   case server_hello_done: ServerHelloDone;
   case certificate_verify: CertificateVerify;
   case client_key_exchange: ClientKeyExchange;
   case finished: Finished; }
   body;
 } Handshake;
```

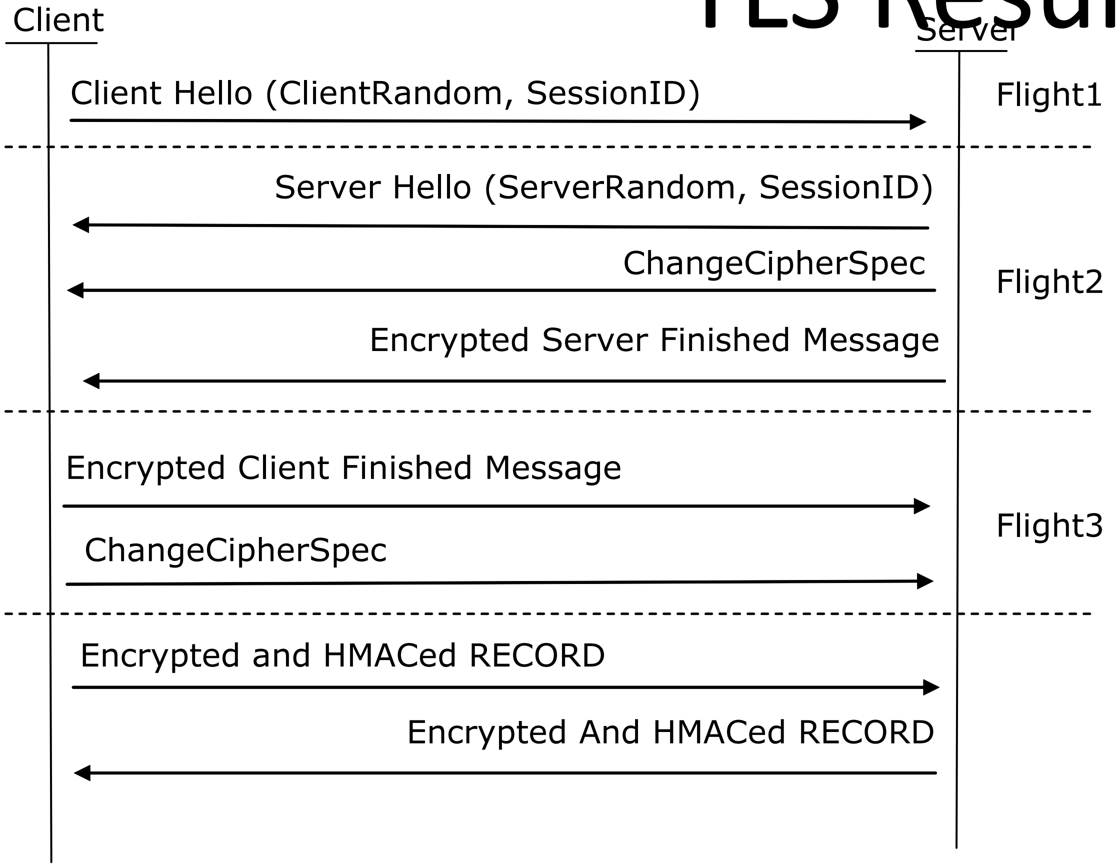
SSL/TLS, Dialogue de base



MasterSecret= PRF(ClientRandom, ServerRandom, PreMasterSecret,…)

Keys = PRF(ClientRandom, ServerRandom, MasterSecret,…)

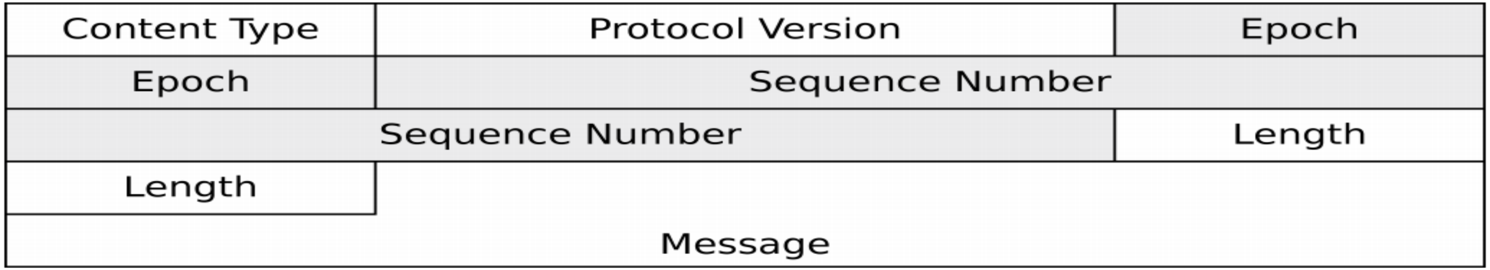
TLS Resume Mode



Keys = PRF(ClientRandom, ServerRandom, MasterSecret,…)

DTLS

DTLS: Record Layer




```
struct {
ContentType type;
ProtocolVersion version;
uint16 epoch;
uint48 sequence-number;
uint16 length;
opaque fragment[DTLSPlaintext.length];
} DTLSPlaintext;
```

epoch: A counter value that is incremented on every cipher state change.

DTLS MAC

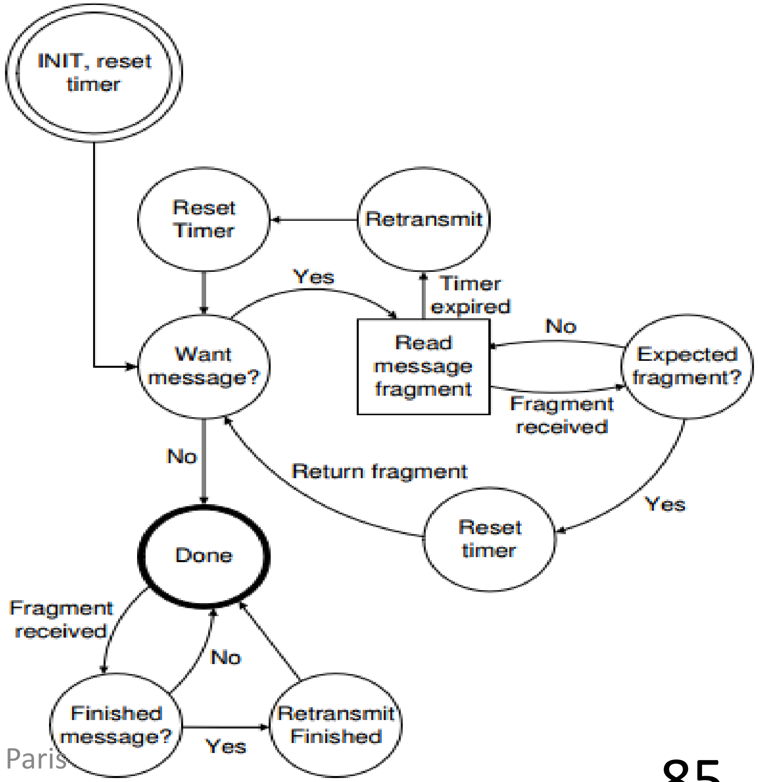
/* The DTLS MAC is the same as that of TLS 1.1. However, rather than
using TLS's implicit sequence number, the sequence number used to
compute the MAC is the 64-bit value formed by concatenating the epoch
and the sequence number in the order they appear on the wire.
Note
that the DTLS epoch + sequence number is the same length as the TLS
sequence number.
*/

HMAC_(Kauth,seq_num + || type || version || length || message)

DTLS: Fragmentation des messages Handshake

- Tous les calculs cryptographiques sont réalisés avec des messages non fragmentés
- Le paramètre *message-sequence* est ignoré

HandshakeType msgtype;
uint24 length;
uint16 message-sequence;
uint24 fragment-offset;
uint24 fragment-length;
[Handshake Message]



Flights DTLS

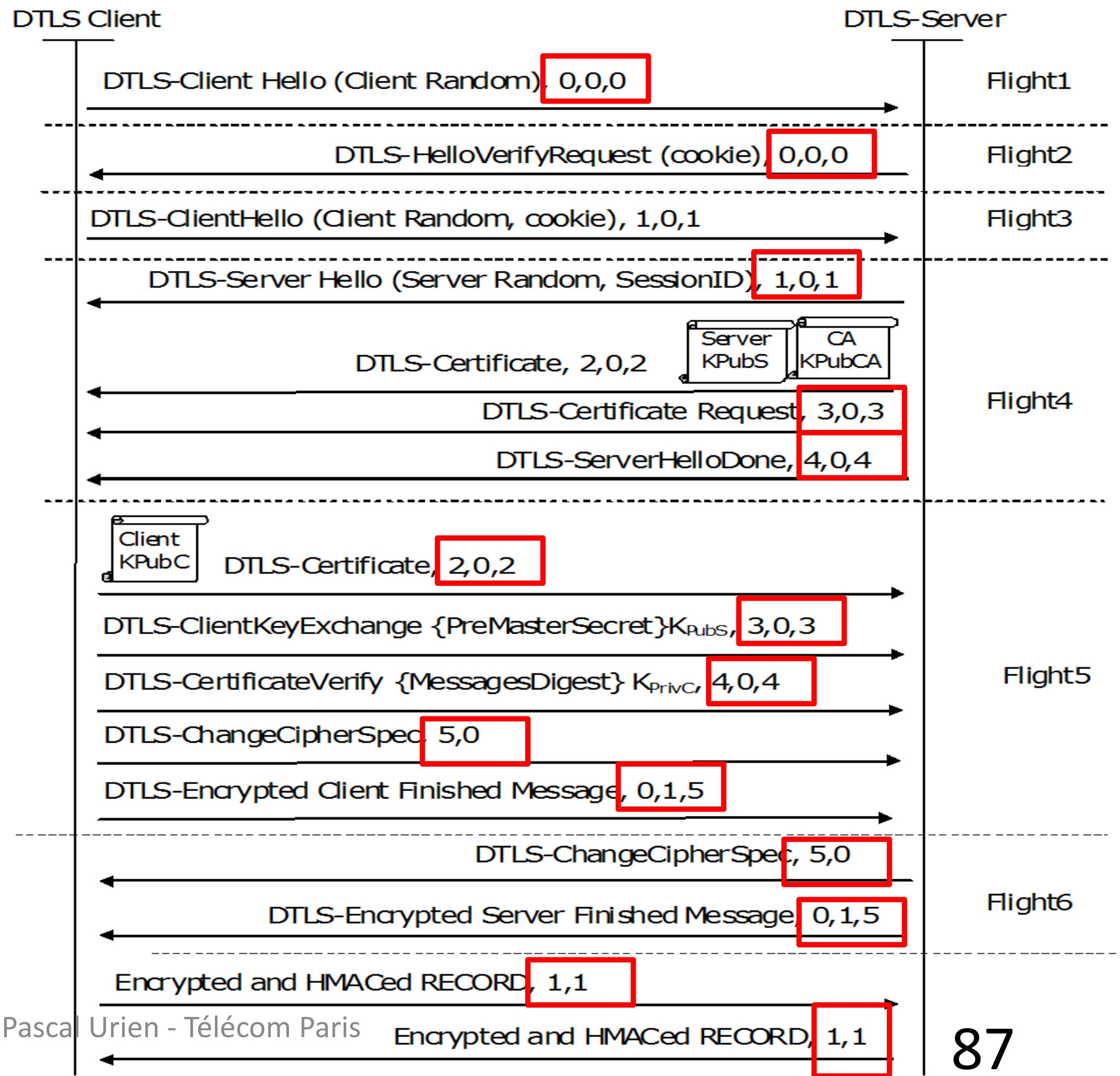
```
  Client                                   Serveur
  ------                                   -------
flight1 ->
(DTLS client-hello)
                                         <- Flight 2
                                       (DTLS Hello-Verify-Request
                                        contains cookie)
 flight 3_>
(DTLS client-hello
 with cookie)
                                        <- flight 4
                                       (DTLS server-hello,
                                        DTLS certificate,
                                        DTLS server-key-exchange,
                                        DTLS certificate-request,
                                        DTLS server-hello-done)
flight 5 ->
(DTLS certificate,
 DTLS client-key-exchange,
 DTLS certificate-verify,
 DTLS change-cipher-spec,
 DTLS finished)
                                       <- flight 6
                                       (DTLS change-cipher-spec,
                                        DTLS finished)
```

DTLS Session Full 1.1

(record sequence number,
epoch,
handshake sequence number)

(record sequence number, epoch)

epoch A counter value that is incremented on every cipher state change.



TLS 1.3

RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3
RFC 8448: Example Handshake Traces for TLS 1.3
RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)

Objectifs

- Canal sure
  - Authenticated encryption with associated data (AEAD)
- Chiffrement de messages handshake et data.
- Perfect forward secrecy (PFS) basé sur des échanges Diffie Hellman (DHE)
- Mise en oeuvre de courbes elliptiques pour les DHE
- Mise en oeuvre de HMAC-based Extract-and-Expand Key Derivation Function (HKDF) pourles dérivations des clés

Record Layer : 17 03 03 Length_MSB Length_LSB AEAD

BASIC Exchange (RT1)

```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                            + pre_shared_key*  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]
```

+ Indicates noteworthy extensions sent in the previously noted message.
* Indicates optional or situation-dependent messages/extensions that are not always sent.
() Indicates messages protected using keys derived from a client_early_traffic_secret.
{} Indicates messages protected using keys derived from a [sender]_handshake_traffic_secret.
[] Indicates messages protected using keys derived from [sender]_application_traffic_secret_N.

RTT0 (pre shared key)

```
Client                                               Server

ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
(Application Data*)     -------->
                                                ServerHello
                                           + pre_shared_key
                                               + key_share*
                                      {EncryptedExtensions}
                                              + early_data*
                                                 {Finished}
                        <--------       [Application Data*]
(EndOfEarlyData)
{Finished}              -------->
[Application Data]      <------->        [Application Data]
```

+ Indicates noteworthy extensions sent in the previously noted message.
* Indicates optional or situation-dependent messages/extensions that are not always sent.
() Indicates messages protected using keys derived from a client_early_traffic_secret.
{} Indicates messages protected using keys derived from a [sender]_handshake_traffic_secret.
[] Indicates messages protected using keys derived from [sender]_application_traffic_secret_N.

TLS1.3 Basic Exchange
rfc 8448

```
16 03 01 00 c4  // Record
01 00 00 c0 // Client Hello (01)
03 03 // Version
// Random
cb 34 ec b1 e7 81 63 ba 1c 38 c6 da cb 19 6a 6d ff a2 1a 8d 99 12 ec 18 a2 ef 62 83 02 4d ec e7
00 // SessionID length
00 06 // Cipher suite length
1301 1303 1302
01 // compression
00 // null
00 91 // Extension length
00 00 00 0b // Server Name
00 09 00 00 06 73 65 72 76 65 72
ff 01 00 01
00
00 0a 00 14 // supported groups, 001d= Curve25519
0012 001d 0017 0018 0019 0100 0101 0102 0103 0104
00 23 00 00 // Tls Ticket
00 33 00 26 // key_share
00 24
001d (group) 00 20   // Diffie Hellman
99 38 1d e5 60 e4 bd 43 d2 3d 8e 43 5a 7d ba fe b3 c0 6e 51 c1 3c ae 4d 54 13 69 1e 52 9a af 2c
00 2b 00 03 02 03 04 // supported_versions
00 0d 00 20 // signature_algorithms
00 1e 04 03 05 03 06 03 02 03 08 04 08 05 08 06 04 01 05 01 06 01 02 01 04 02 05 02 06 02 02 02
00 2d 00 02 01 01 // psk_key_exchange_mode= 1 = psk_dhe_ke.
00 1c 00 02 40 01 // record_size_limit
```

```
ServerHello (90 octets)
16 03 03 00 5a
02 00 00 56 // Server Hello (2)
03 03 // Version
// Random
a6 af 06 a4 12 18 60 dc 5e 6e 60 24 9c d3 4c
95 93 0c 8a c5 cb 14 34 da c1 55 77 2e d3 e2
69 28
00 // SessionID Length
13 01 // CipherSuite
00 // Compression Method (null)
00 2e // Extensions Length
00 33 00 24 // key_share extension
00 1d 00 20
c9 82 88 76 11 20 95 fe 66 76 2b db f7 c6 72
e1 56 d6 cc 25 3b 83 3d f1 dd 69 b1 b0 4e 75
1f 0f 00 2b 00 02 03 04 // record_size_limit
```

Cryptographic Calculations

- DHE calculation according to the Elliptic Curve
- HMAC-based Extract-and-Expand Key Derivation Function (HKDF), RFC 6869
  - PRK = HMAC-Hash(salt, IKM) , IKM=Input Key Material
  - OKM (Output Key Material)
    - T(0) = empty string (zero length)
    - T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)
    - T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)
    - T(k) = HMAC-Hash(PRK, T(k-1) | info | 0x0k)

- The key derivation process makes use of the **HKDF-Extract and HKDF-Expand** functions as defined for HKDF [RFC5869], as well as the functions defined below:

  HKDF-Expand-Label(Secret, Label, Context, Length) = **HKDF-Expand**(Secret, **HkdfLabel**, Length)

- Where HkdfLabel is specified as:

```
struct { uint16 length = Length;
         opaque label<7..255> = "tls13 " + Label;
         opaque context<0..255> = Context; }  HkdfLabel;
```

- ***Derive-Secret***(Secret, Label, Messages) = ***HKDF-Expand-Label***(Secret, Label, Transcript-Hash(Messages), Hash.length)
  - The Hash function used by Transcript-Hash and HKDF is the cipher suite hash algorithm.

Keys are derived from two input secrets using the HKDF-Extract and Derive-Secret functions.

The general pattern for adding a new secret is to use HKDF-Extract with the Salt being the current secret state and the Input Keying Material (IKM) being the new secret to be added.

In this version of TLS 1.3, the two input secrets are:
-PSK (a pre-shared key established externally or derived from the resumption_master_secret value from a previous connection)
-(EC)DHE shared secret.

This produces a full key derivation schedule shown in the diagram below. In this diagram, the following formatting conventions apply:
-HKDF-Extract is drawn as taking the Salt argument from the top and the IKM argument from the left, with its output to the bottom and the name of the output on the right.
-Derive-Secret's Secret argument is indicated by the incoming arrow. For instance, the Early Secret is the Secret for generating the client_early_traffic_secret.
- "0" indicates a string of Hash.length bytes set to zero.

Pre shared Key

```
         SALT
IKM       0
          |
          v
PSK ->  HKDF-Extract = Early Secret
          |
          +-----> Derive-Secret(., "ext binder" | "res binder", "")
          |                     = binder_key
          |
          +-----> Derive-Secret(., "c e traffic", ClientHello)
          |                     = client_early_traffic_secret
          |
          +-----> Derive-Secret(., "e exp master", ClientHello)
          |                     = early_exporter_master_secret
          v
    Derive-Secret(., "derived", "")
```

IKM SALT DERIVED SECRET

```
(EC)DHE  -> HKDF-Extract = Handshake Secret
             |
             +-----> Derive-Secret(., "c hs traffic",
             |                     ClientHello...ServerHello)
             |                     = client_handshake_traffic_secret
             |
             +-----> Derive-Secret(., "s hs traffic",
             |                     ClientHello...ServerHello)
             |                     = server_handshake_traffic_secret
             v
       Derive-Secret(., "derived", "")
             |
             v
   0 -> HKDF-Extract = Master Secret
             |
             +-----> Derive-Secret(., "c ap traffic",
             |                     ClientHello...server Finished)
             |                     = client_application_traffic_secret_0
             |
             +-----> Derive-Secret(., "s ap traffic",
             |                     ClientHello...server Finished)
             |                     = server_application_traffic_secret_0
             |
             +-----> Derive-Secret(., "exp master",
             |                     ClientHello...server Finished)
             |                     = exporter_master_secret
             |
             +-----> Derive-Secret(., "res master",
                                   ClientHello...client Finished)
                                   = resumption_master_secret
```

Hi

1) Traditionally, a HKDF-Extract is used to extract entropy from a DH type shared secret. However, the first HKDF-Extract in the key schedule takes a PSK instead of a DH shared secret.
We don't see security problems with this instance in TLS 1.3. NIST requires the PSK to have efficient amount of entropy (to achieve a security strength required by NIST) when it is externally generated. When it is externally generated, one of NIST's approved random bit generation methods in SP 800-90 series must be used.
When the PSK is a resumption key, then its original key exchange and its key derivation function(s) must meet the security strength desired/required for the PSK.

Traffic Keys

- Once the handshake is complete, it is possible for either side to update its sending traffic keys using the ***KeyUpdate handshake message***
  - application_traffic_secret_N+1 = HKDF-Expand-Label (application_traffic_secret_N, "traffic upd", "", Hash.length)
- The traffic keying material is generated from an input traffic secret value using:
  - [sender]_write_key = HKDF-Expand-Label(Secret, "key", "", key_length)
  - [sender]_write_iv = HKDF-Expand-Label(Secret, "iv", "", iv_length)

```
EncryptedExtensions (40 octets):
08 00 00 24
00 22 00 0a 00 14 00 12 00 1d 00 17 00 18 00 19 01 00 01 01 01
02 01 03 01 04 00 1c 00 02 40 01 00 00 00 00
Certificate (445 octets):
0b 00 01 b9
00 00 01 b5
00 01 b0
30 82 01 ac ...
CertificateVerify (136 octets):
0f 00 00 84
...
Finished (36 octets):
14 00 00 20
9b 9b 14 1d 90 63 37 fb d2 cb dc e7 1d f4 de da 4a b4 2c 30 95
72 cb 7f ff ee 54 54 b7 8f 07 18

Encrypted Record
Message complete Encrypted record (679 octets):
17 03 03 02 a2
d1 ff 33 4a 56 f5 bf ...
```

```
Client Finished (36 octets):
14 00 00 20
a8 ec 43 6d 67 76 34 ae 52 5a c1 fc eb e1 1a 03 9e c1
76 94 fa c6 e9 85 27 b6 42 f2 ed d5 ce 61

Complete Encrypted Record (58 octets):
17 03 03 00 35
00 f8 b4 67 d1 4c f2 2a 4b 3f 0b 6a e0 d8 e6 cc 8d 08
e0 db 35 15 ef 5c 2b df 19 22 ea fb b7 00 09 96 47 16
d8 34 fb 70 c3 d2 a5 6c 5b 1f 5f 6b db a6 c3 33 cf
```

RT0 - Pre Shared Key

Client Hello

```
16 03 01 02 00 // Record
01 00 01 fc // Client Hello(01)
03 03 // Version
1b c3 ce b6 bb e3 9c ff 93 83 55 b5 a5 0a db 6d // Client Random
b2 1b 7a 6a f6 49 d7 b4 bc 41 9d 78 76 48 7d 95
00 // SessionID length
00 06 // CipherID Length
1301 1303 1302
01 // Compression
00 // Null
01 cd // Extensions Length
00 00 00 0b // Server Name
00 09 00 00 06 73 65 72 76 65 72
ff 01 00 01 //
00
00 0a 00 14 // suported groups, 001d= Curve25519
0012 001d 0017 0018 0019 0100 0101 0102 0103 0104
00 33 00 26 // key_share
00 24 00 1d 00 20 e4 ff b6 8a c0 5f 8d 96 c9 9d
a2 66 98 34 6c 6b e1 64 82 ba dd da fe 05 1a 66
b4 f1 8d 66 8f 0b
00 2a 00 00   // early_data
00 2b 00 03   // supported_versions
02 03 04
```

```
00 0d 00 20 // signature_algorithms
00 1e 04 03 05 03 06 03 02 03 08 04 08 05 08 06
04 01 05 01 06 01 02 01 04 02 05 02 06 02 02 02
00 2d 00 02 // psk_key_exchange_modes
01 01 // psk_dhe_ke.
00 1c 00 02 // record_size_limit
40 01
00 15 00 57 // 21=Padding
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00
00 29 00 dd // 41=pre_shared_key
00 b8 // PskIdentity
00 b2 // opaque_identity
2c 03 5d 82 93 59 ee 5f f7 af 4e c9 00 00 00 00 26 2a 64 94 dc 48 6d 2c 8a 34 cb 33 fa 90 bf 1b
00 70 ad 3c 49 88 83 c9 36 7c 09 a2 be 78 5a bc 55 cd 22 60 97 a3 a9 82 11 72 83 f8 2a 03 a1 43
ef d3 ff 5d d3 6d 64 e8 61 be 7f d6 1d 28 27 db 27 9c ce 14 50 77 d4 54 a3 66 4d 4e 6d a4 d2 9e
e0 37 25 a6 a4 da fc d0 fc 67 d2 ae a7 05 29 51 3e 3d a2 67 7f a5 90 6c 5b 3f 7d 8f 92 f2 28 bd
a4 0d da 72 14 70 f9 fb f2 97 b5 ae a6 17 64 6f ac 5c 03 27 2e 97 07 27 c6 21 a7 91 41 ef 5f 7d
e6 50 5e 5b fb c3 88 e9 33 43 69 40 93 93 4a e4 d3 57
fa d6 aa cb //obfuscated_ticket_age
00 21 // opaque PskBinderEntry
20 // HMAC value
3a dd 4f b2 d8 fd f8 22 a0 ca 3c f7 67 8e f5 e8 8d ae 99 01 41 c5 92 4d 57 bb 6f a3 1b 9e 5f 9d
```

Client Hello

```
16 03 03 00 60 // Record
02 00 00 5c // Server Hello (02)
03 03 // Version
3c cf d2 de c8 90 22 27 63 47 2a e8 13 67 77 c9 // Server Random
d7 35 87 77 bb 66 e9 1e a5 12 24 95 f5 59 ea 2d
00 // SessionID Length
13 01 // Cipher suite
00 // Compression Method (null)
00 34 // Extension List
00 29 00 02  // Pre-shared-key
00 00
00 33 00 24 // 51=key_share
00 1d 00 20 // Curve25519
12 17 61 ee 42 c3 33 e1 b9 e7 7b 60 dd 57 c2 05 3c d9 45 12 ab 47 f1 15 e8 6e
ff 50 94 2c ea 31 00 2b 00 02 03 04
```

Server Hello

```
payload (80 octets):
08 00 00 28 // Encrypted Extensions
00 26
00 0a 00 14 // supported_groups
0012 001d 0017 0018 0019 0100 0101 0102 0103 0104
00 1c 00 02 //
40 01
00 00 00 00 // Server Name
00 2a 00 00 // early_data

14 00 00 20 // Server Finished
48 d3 e0 e1 b3 d9 07 c6 ac ff 14 5e 16 09 03 88
c7 7b 05 c0 50 b6 34 ab 1a 88 bb d0 dd 1a 34 b2

Encrypted Record Message
17 03 03 00 61
dc 48 23 7b 4b 87 9f 50 d0 d4 d2 62 ea 8b 47 16
eb 40 dd c1 eb 95 7e 11 12 6e 8a 71 49 c2 d0 12
d3 7a 71 15 95 7e 64 ce 30 00 8b 9e 03 23 f2 c0
5a 9c 1c 77 b4 f3 78 49 a6 95 ab 25 50 60 a3 3f
ee 77 0c a9 5c b8 48 6b fd 08 43 b8 70 24 86 5c
a3 5c c4 1c 4e 51 5c 64 dc b1 36 9f 98 63 5b c7
a5
```

Server Encrypted Extensions + finished

```
Client Finished (36 octets):
14 00 00 20
72 30 a9 c9 52 c2 5c d6 13 8f c5 e6 62 83 08 c4
1c 53 35 dd 81 b9 f9 6b ce a5 0f d3 2b da 41 6d

Client Encrypted Record (58 octets):
17 03 03 00 35
00 f8 b4 67 d1 4c f2 2a 4b 3f 0b 6a e0 d8 e6 cc
8d 08 e0 db 35 15 ef 5c 2b df 19 22 ea fb b7 00
09 96 47 16 d8 34 fb 70 c3 d2 a5 6c 5b 1f 5f 6b
db a6 c3 33 cf
```

Client
Encrypted Finished

```
Extension; enum {
server_name(0), /* RFC 6066 */
max_fragment_length(1), /* RFC 6066 */
status_request(5), /* RFC 6066 */
supported_groups(10), /* RFC 8422, 7919 */
signature_algorithms(13), /* RFC 8446 */ u
se_srtp(14), /* RFC 5764 */
heartbeat(15), /* RFC 6520 */
application_layer_protocol_negotiation(16), /* RFC 7301 */
signed_certificate_timestamp(18), /* RFC 6962 */
client_certificate_type(19), /* RFC 7250 */
server_certificate_type(20), /* RFC 7250 */
padding(21), /* RFC 7685 */
pre_shared_key(41), /* RFC 8446 */
early_data(42), /* RFC 8446 */
supported_versions(43), /* RFC 8446 */
cookie(44), /* RFC 8446 */
psk_key_exchange_modes(45), /* RFC 8446 */
certificate_authorities(47), /* RFC 8446 */
oid_filters(48), /* RFC 8446 */
post_handshake_auth(49), /* RFC 8446 */
signature_algorithms_cert(50), /* RFC 8446 */
key_share(51), /* RFC 8446 */
(65535)
} ExtensionType;
```

```
struct
{ ExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
}
```