第8章 IPSec 与 SSL

南京大学计算机系黄皓教授

2009年12月

参考文献

I. RFC2401: Security Architecture for the Internet Protocol.

II. RFC2402: IP Authentication Header.

III. RFC2403: The Use of HMAC-MD5-96 within ESP and AH.

IV. RFC2404: The Use of HMAC-SHA-1-96 within ESP and AH.

V. RFC2405: The ESP DES-CBC Cipher Algorithm With Explicit IV.

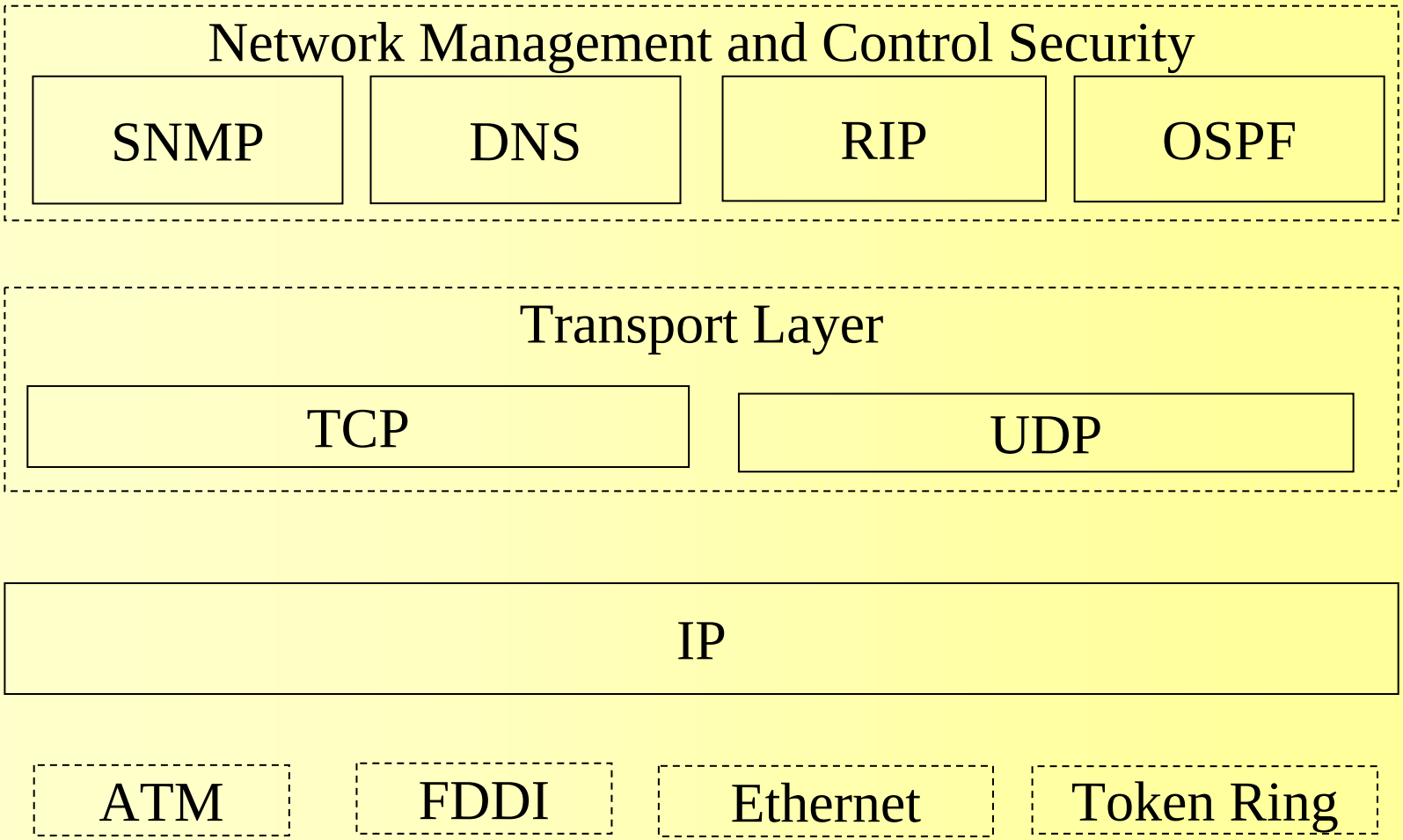
- RFC2406: IP Encapsulating Security Payload (ESP).
- RFC2407: The Internet IP Security Domain of Interpretation for ISAKMP.
- RFC2408: The Internet IP Security Domain of Interpretation for ISAKMP.
- RFC2409: The Internet Key Exchange (IKE).
- RFC2410: The NULL Encryption Algorithm and Its Use With Ipsec.
- RFC2411: IP Security Document Roadmap.
- RFC2412: The OAKLEY Key Determination Protocol。
- William Stallings, 密码编码学与网络安全-原理与实践，第13章。

内容

1. 引言
2. 安全关联
3. 认证头
4. 安全载荷封装
5. IKE

1. introduction

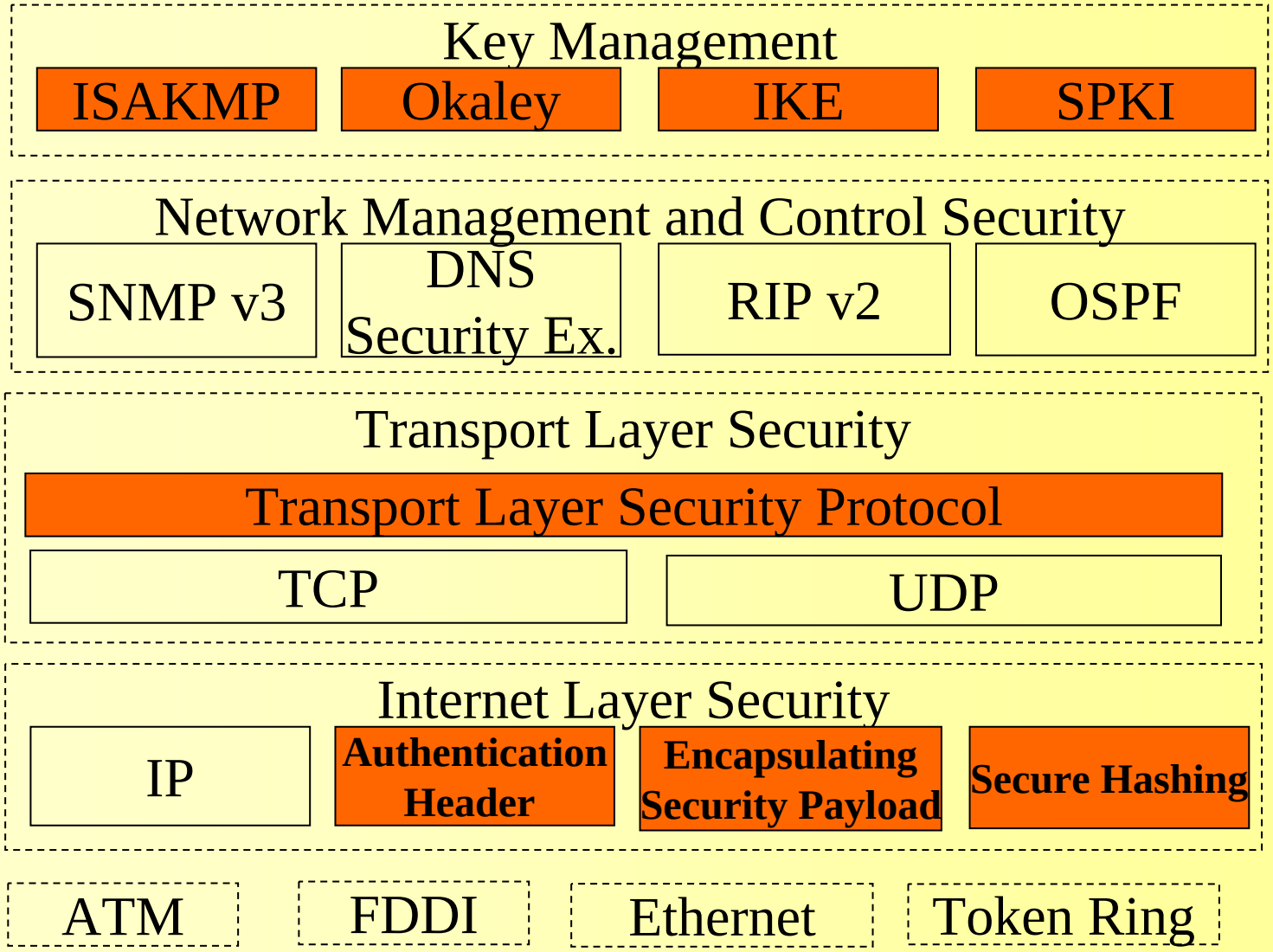
TCP/IP Protocols



IP数据包不具有安全性

- 攻击者可以:
  - 伪造IP包的地址
  - 查看IP包的内容
  - 重播以前的包
  - 修改IP包的内容
- 通信者不能:
  - 保证IP包的真实来源
  - 保证IP包没有泄密
  - IP包发送者的当前意图
  - 发送者的真实数据

Internet 安全协议栈



2. System overview

什么是IPsec?

- IPsec 在 IP层提供安全服务
  - 选择安全协议
  - 选择安全协议中要使用大的密码算法
  - 管理密钥
- 保护一个或几个安全信道
  - 两个主机之间的安全信道
  - 两个网关之间的安全信道;
  - 一个主机和一个网关之间的安全信道

IPsec提供的安全服务

- 访问控制
- 无连接通信的完整性
- 数据源的验证
- 拒绝重播的数据包
- 机密性
- 有限的通信量的机密性
- 直接为上层协议服务： TCP, UDP,ICMP, BGP等。

IPsec的组成

- 安全协议
  - Authentication Header (AH)
  - Encapsulating Security Payload (ESP)
- 自动密钥协商
  - The Internet Key Exchange (IKE)
  - Internet Security Association and Key Management Protocol (ISAKMP)
  - The OAKLEY Key Determination Protocol
  - The Internet IP Security Domain of Interpretation for ISAKMP

IPsec 如何工作?

- IPsec利用两个安全协议 (AH,ESP)提供安全服务
  - AH 提供无连接运输的完整性、数据源验证、抗重播
  - ESP机密性、通信量保密、无连接运输的完整性、数据源验证、抗重播
  - AH 和ESP 单独或者联合使用

IP的实现模式

- 两种模式
  - 运输模式(transport mode)　为上层协议提供安全服务

IP head	AH head	ESP head	payload

  - 隧道模式(tunnel mode)　封装IP报文;

IP head	**AH head**	**ESP head**	**IP head**	**payload**

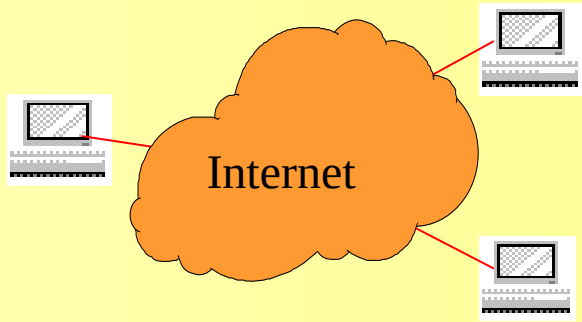
安全信道的粒度

- 一个安全信道可以在两个网关之间保护所有的通信数据
- 保护两个网关之间的某一类网络服务的通信数据
- 保护某一个网络会话

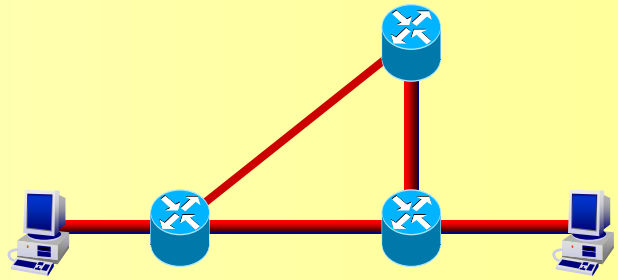
- IPsec的管理机制
  - 采用哪些安全服务设施
  - 选择安全信道的粒度;
  - 选择密码算法;

IPsec 的部署

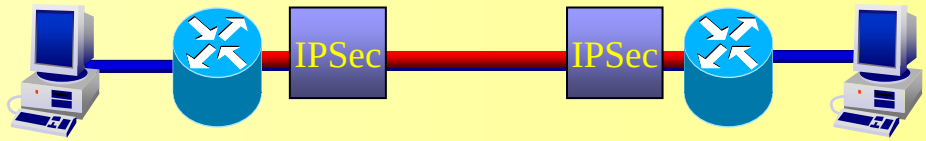
- 在一个主机上



- 在网关上

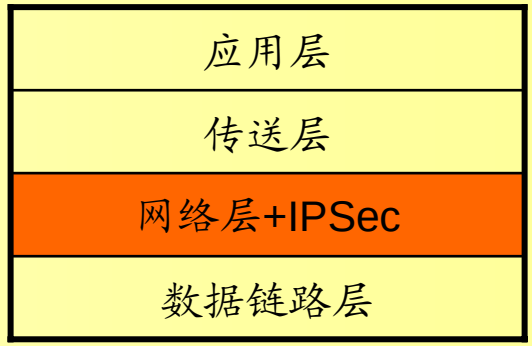


- 在线路中

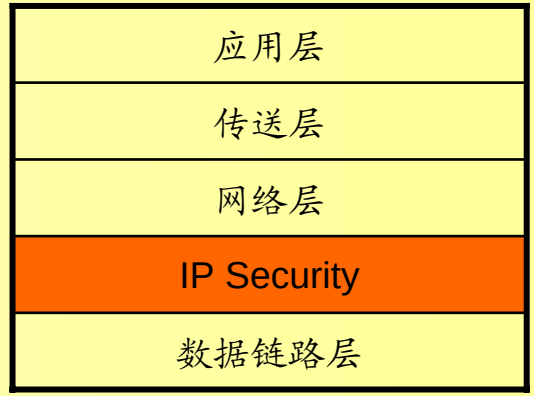


IPsec 的实现方法

- 与本地的协议栈的IP结合



- 增加协议层



3. 安全关联

Security Associations

- AH 和 ESP 都要利用安全关联进行工作
- IKE的主要任务就是建立和管理安全关联

定义

- 安全关联SA是两个通信实体建立起来的“连接”，它们用来保护数据的一系列参数：IPsec的协议类型、加密算法、认证方式、加密和认证的密钥。密钥的生成时间和防重播的序列号等。
- SA可以手工建立。也可以利用密钥交换协议自动建立
- SA是单向的
  - 如果主机A和主机B利用ESP通信，则主机A的SA(out)和主机B(in)共享相同的”连接”，主机A的SA(in)和主机B(out)共享另外一个相同的”连接”
- 协议相关性
  - SA可以用于AH，也可以用于ESP；但是不能同时用于AH和ESP两个协议。

SA 参数

- 序列号
  - 32位数用来AH和ESP的头部序列号域，在一个安全信道的发出报文时填充。
- 序列号溢出处理标志位
  - 当序列号溢出时是否要产生一个审计事件并禁止利用这个SA进一步传输数据

SA 参数

- 防重播的窗口
- AH 相关的参数
  - 认证算法、密钥、密钥生存期
- ESP 相关参数
  - 加密和认证算法
  - 密钥、初始化向量
- SA 的生存期
- IPsec的模式
- 路径MTU

SAD

- 名义安全关联数据库 (SAD)
  - 每一个记录表示一个安全关联
- 出发处理
  - 网络连接－>安全策略数据库的记录－>安全信道的SA
  - 如果适合当前连接的安全策略库中相应的记录中没有指出相应的SA的记录，则协议发起产生相关SA的过程。
- 接收处理
  - SAD中的每一个记录位置由<目标IP 地址, IPsec 协议类型, SPI>确定。

SA 索引

- SA 可以由以下三个元素唯一地确定
  - 安全参数索引 (SPI);
  - 目标IP 地址;
  - 安全协议 (AH or ESP) ;

安全策略数据库(SPD)

The Security Policy Database

- 安全策略数据库
  - SPD 指定给 IP 数据流提供的安全服务
    - 根据源地址、目标地址、数据流的网络协议th等确定。
  - 选择子： 将IP包头和上层协议的包头域映射到SPDB中的一条策略。

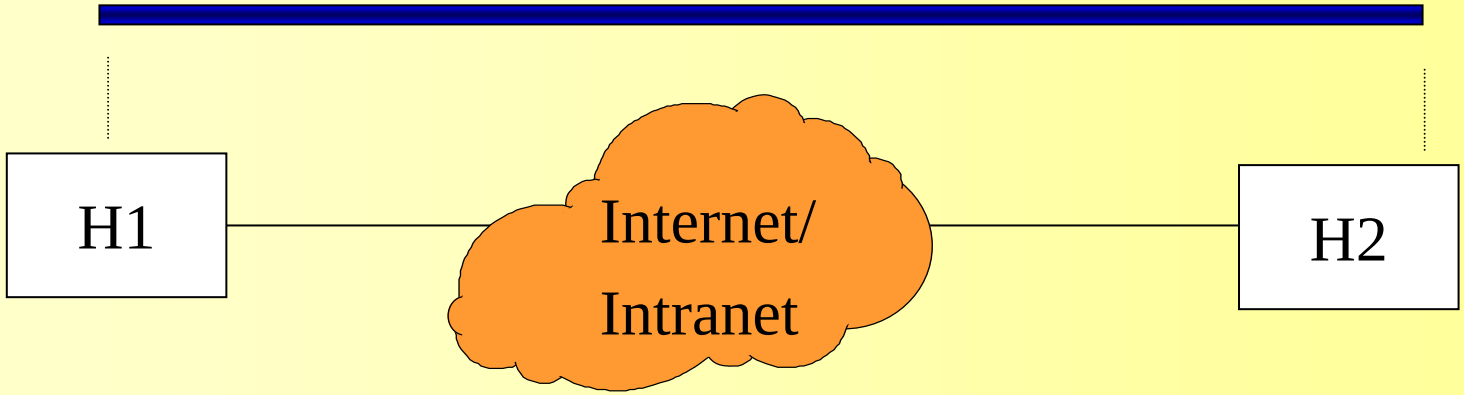
SPD

- 在发送和接收数据的时候都需要检索SPD，以确定数据处理的策略;
- 接收和发送的策略需要分别存储在SPD的不同的策略记录中;
- 在不同的接口上，需要建立不同的SPD;
- SPD对所有的数据流都必须给出策略;
- SPD对一个数据流有三种可能的策略
  - 丢弃、绕过、实施IPsec处理

选择子 (Selector)

- 一个给定的SA有一个确定的“粒度”，其“粒度”由选择子确定，选择子决定哪些数据流由哪个SA处理；
- 选择子由以下因素确定：
  - 目标IP地址
  - 源IP地址
  - 名字
    - 用户名；例如邮件地址mozart@foo.bar.com
    - 系统名称：例如域名等。
  - 安全等级运输层协议
  - 源端口和目的端口

SA组合(1)

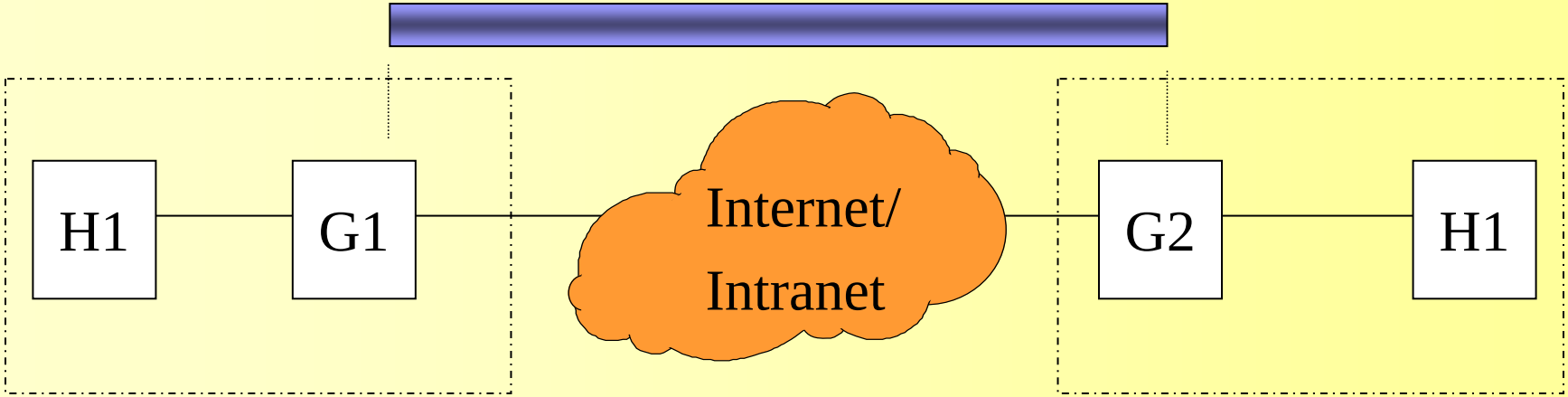


- 两个主机之间

运输模式
1. [IP1][AH][upper]
2. [IP1][ESP][upper]
3. [IP1][AH][ESP][upper]

隧道模式
4. [IP2][AH][IP1][upper]
5. [IP2][ESP][IP1][upper]

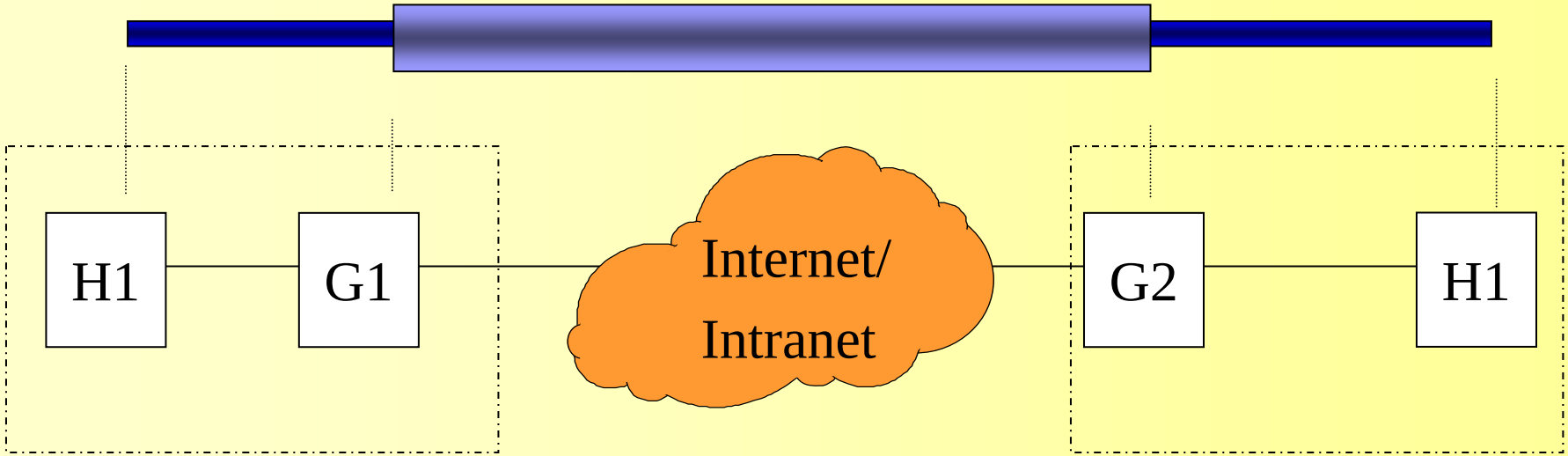
SA组合(2)



- 两个网关之间

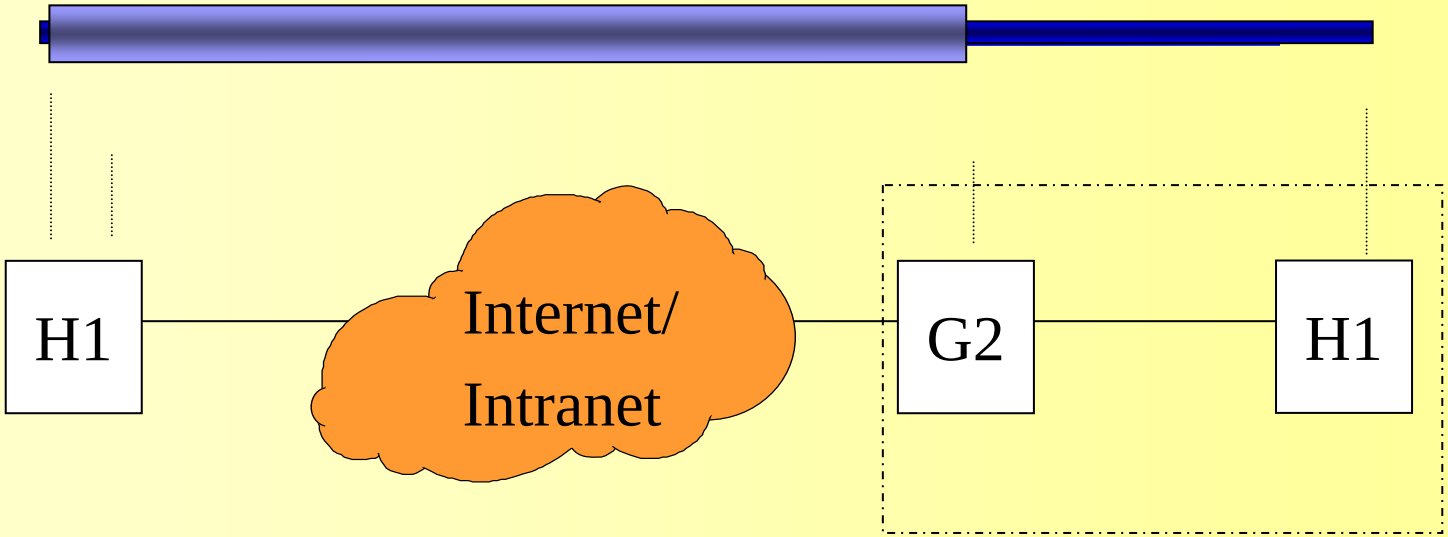
隧道模式
1. [IP2][AH][IP1][upper]
2. [IP2][ESP][IP1][upper]

SA组合(3)



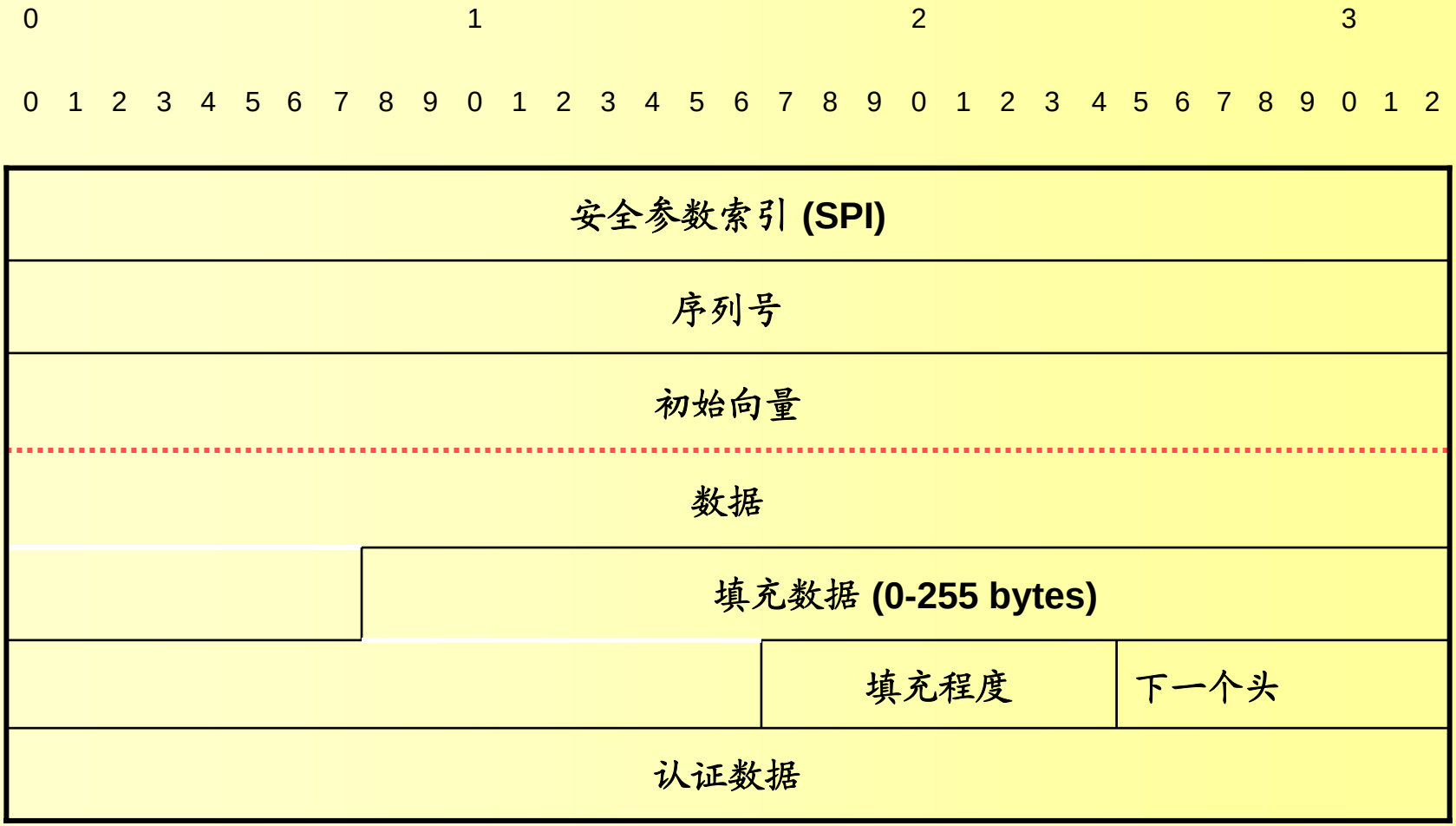
- 主机到主机的信道嵌套在网关到网关之间的信道中。

SA组合(4)



- 主机到网关和网关到网关。

4. 安全载荷封装ESP


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 2 3 4 5 6 7 8 9 0 1 2
安全参数索引 (SPI)
序列号
初始向量
数据
填充数据 (0-255 bytes)
填充程度
下一个头
认证数据

(1) 安全参数索引SPI

- 32位
- 1到255 由 IANA保留
- 由安全信道的接收方在建立SA时顺序;

(2) 序列号

- 32位的无符号整数，表示单调增加的计数
- 收发两端的计数器都从0开始计数.

(3) 载荷数据

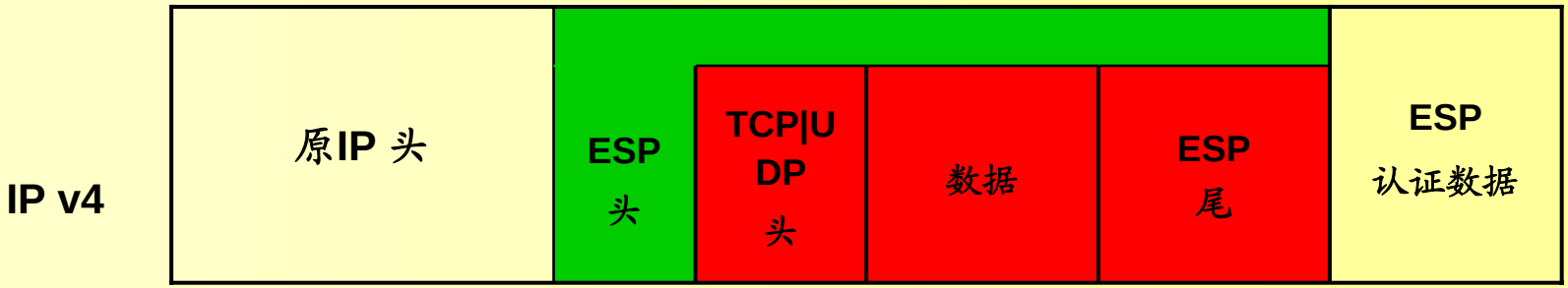
- 由Next Header 域指定的协议类型的数据，长度可变。
- 如果加密模式需要初始向量IV，则IV显式地出现在ESP的数据域中

(4) 填充

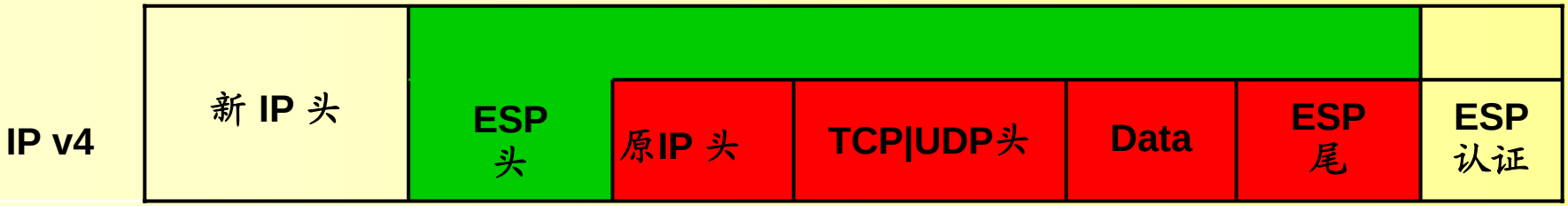
- 填充域用来填充明文域（包括载荷数据、填充长度、下一个头和填充数据），以满足加密算法的需要。
- 填充数据也可以用来隐藏数据的真实长度，支持通信量的保密;

ESP 头的位置

运输模式： ESP处理之后

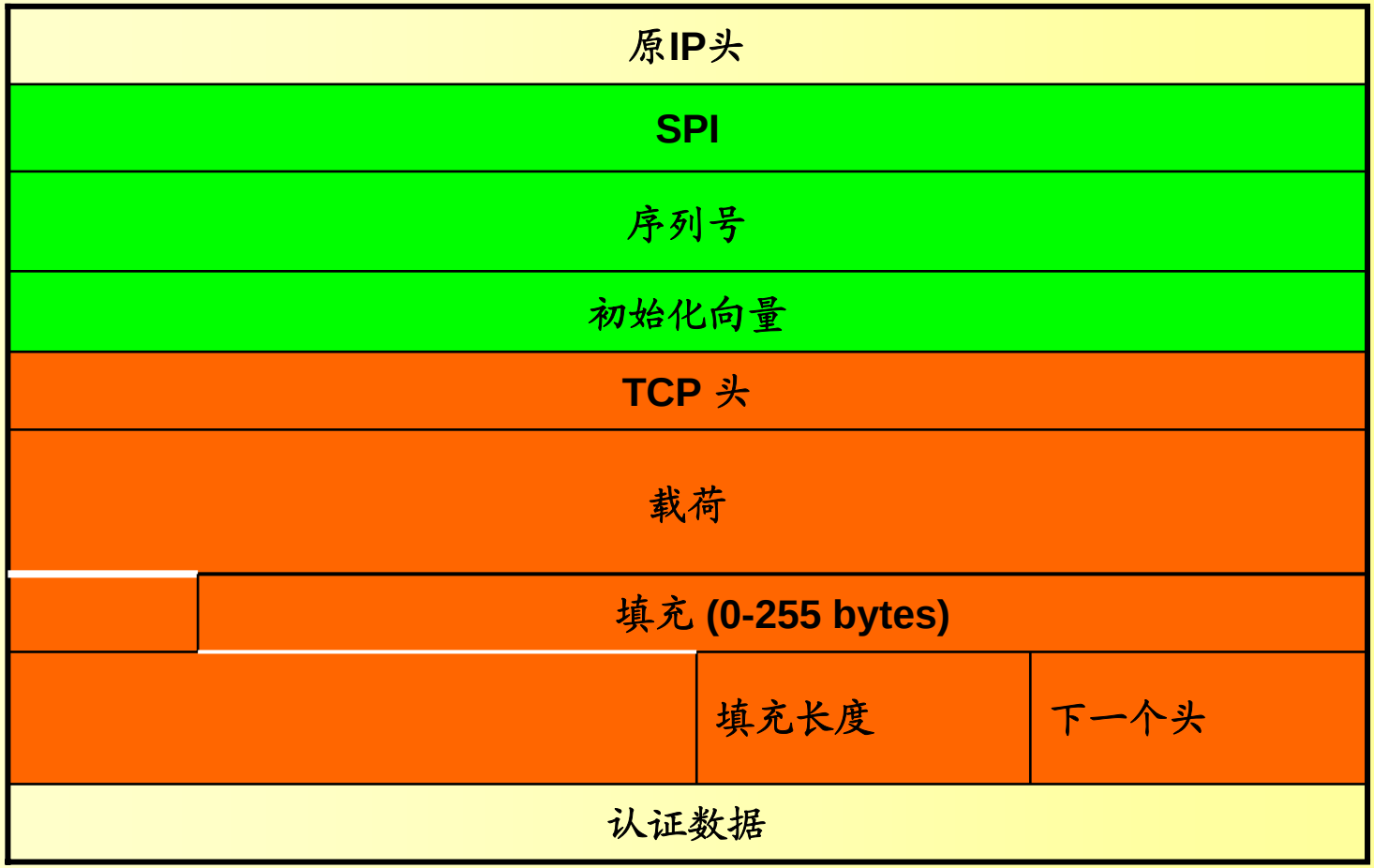


隧道模式 ESP处理之后



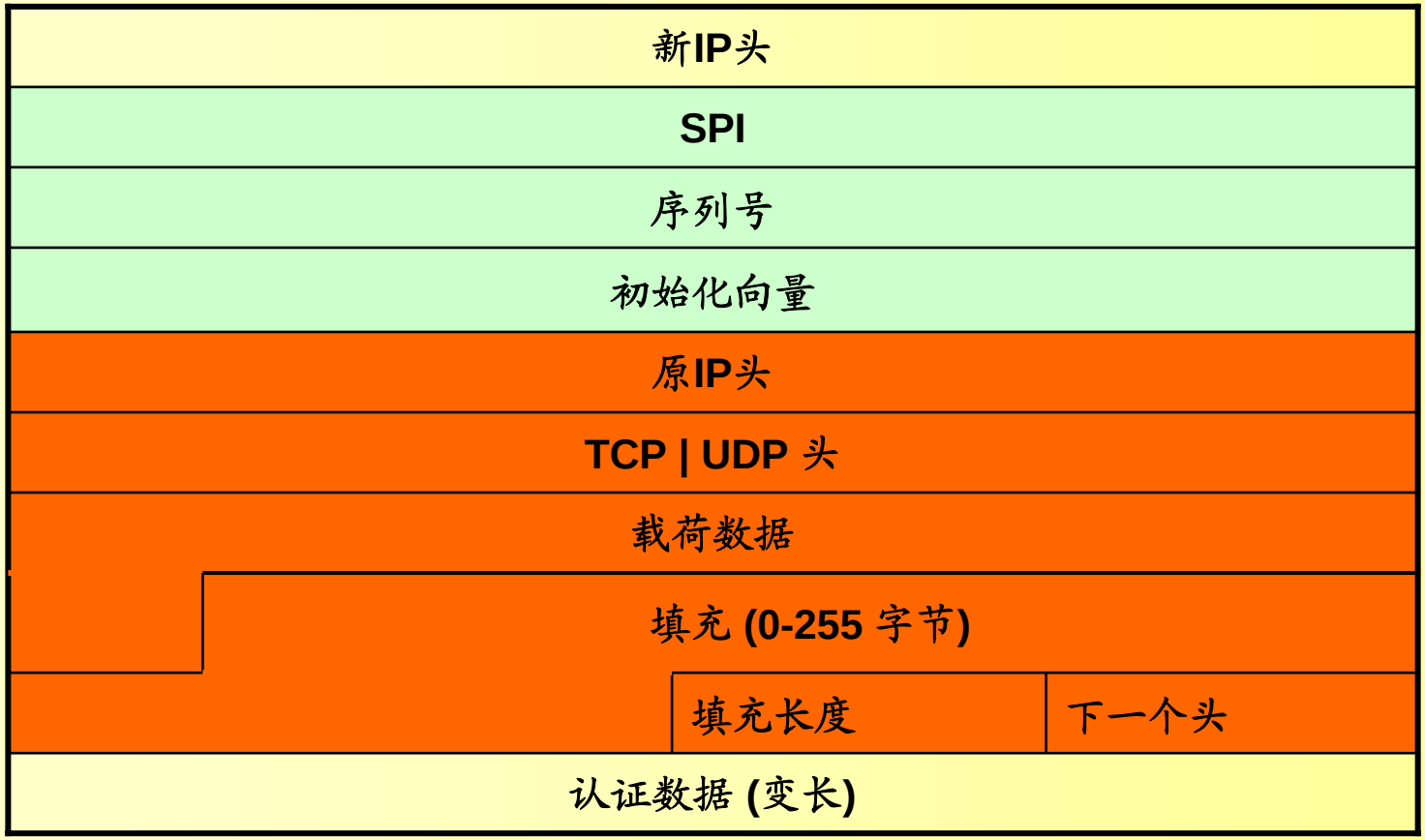
ESP 安全载荷封装

运输模式： ESP处理之后



ESP 安全载荷封装

隧道模式： ESP处理之后



出发包处理

- 安全关联查找
  - 对每一个出发的包，首先检查SPD，找到处理这个包的相关策略。
  - 如果需要丢弃，则记录这个丢弃事件。
  - 如果策略要求绕过IPsec 的处理，则这个报文继续“正常”协议处理。
  - 如果需要进行 IPsec 处理，则将连接映射到一个存在的SA，或激活一个SA的 协商过程。

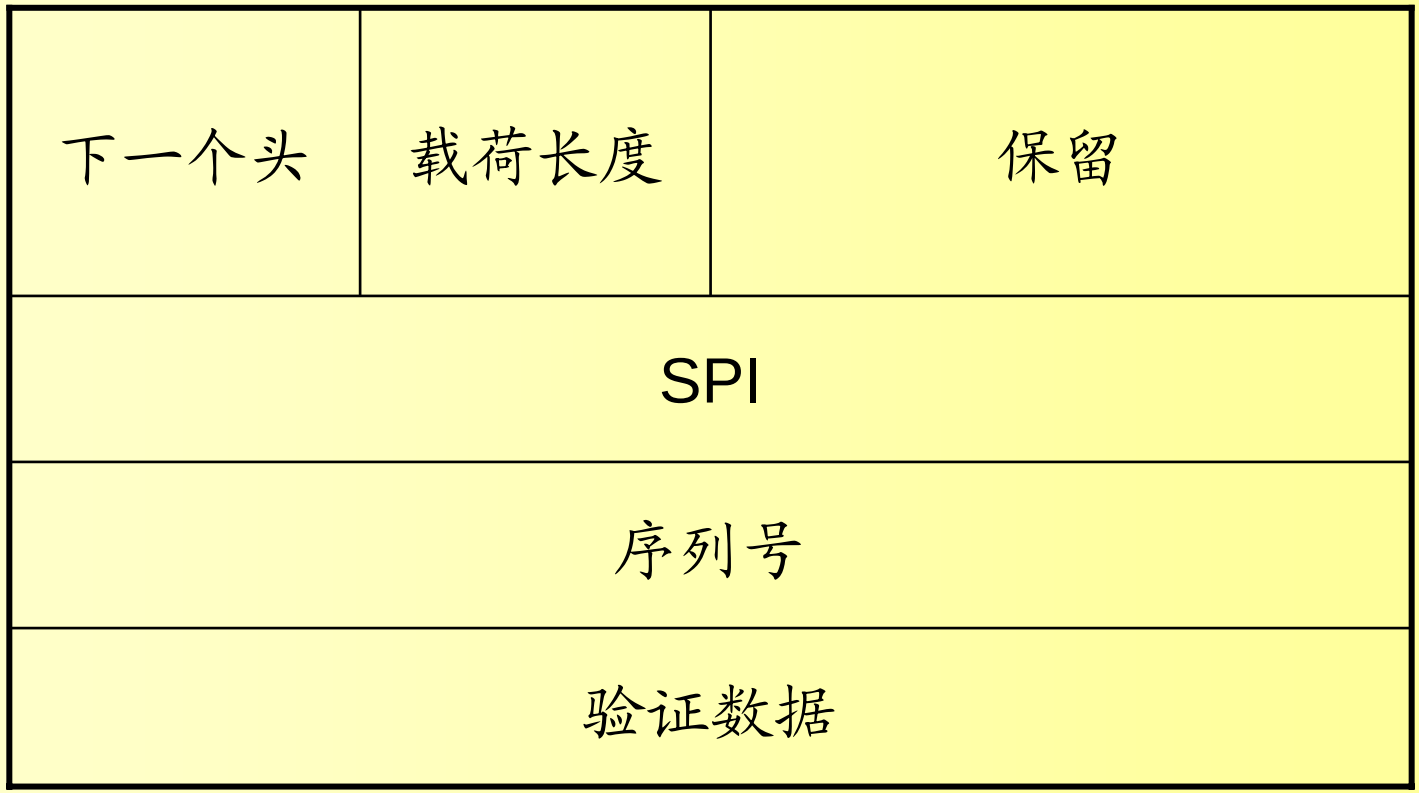
出发包处理

- 加密报文
- 产生序列号
- 完整性验证码产生
- 分片处理

接收包处理

- 组装报文
- 安全关联查找
- 序列号验证
- 完整性检查
- 报文解密

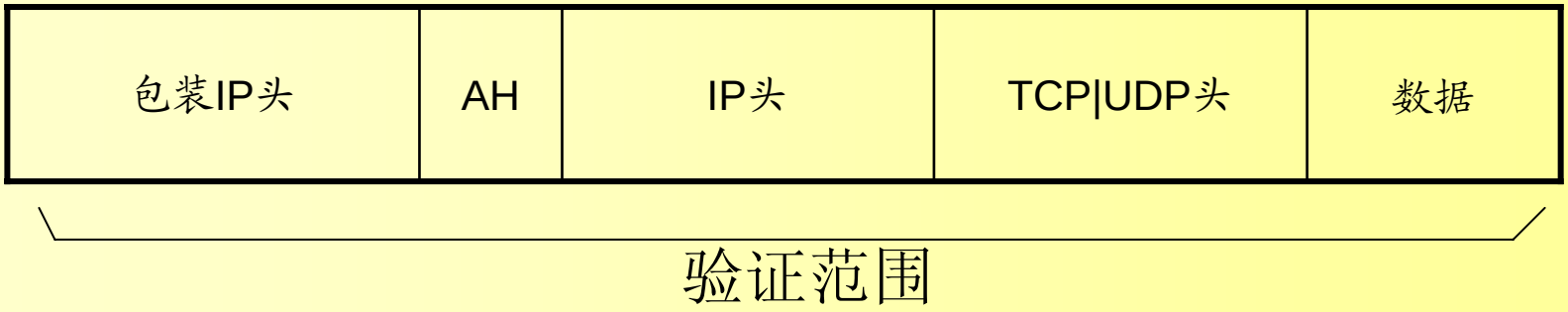
5. 认证头 (AH)



AH 头的位置

- 运输模式 AH

- 隧道模式 AH



出发包处理

- 序列号产生
- 完整性验证数据产生
- 分片

接收包处理

- 组装
- 安全关联查找
- 序列号验证
- 完整性验证

对IPSec的讨论

- 协议灵活性与复杂性
- AH和ESP两个协议同时并存
- 加密和验证的次序
- SA的单向性

6. ISAKMP

ISAKMP

RFC2408:Internet Security Association and Key Management Protocol(ISAKMAP)。

ISAKMP 的目标

- 定义通信双方协商密钥的消息
- 协商的内容:
  - 密码算法
  - 认证机制
  - 密钥交换

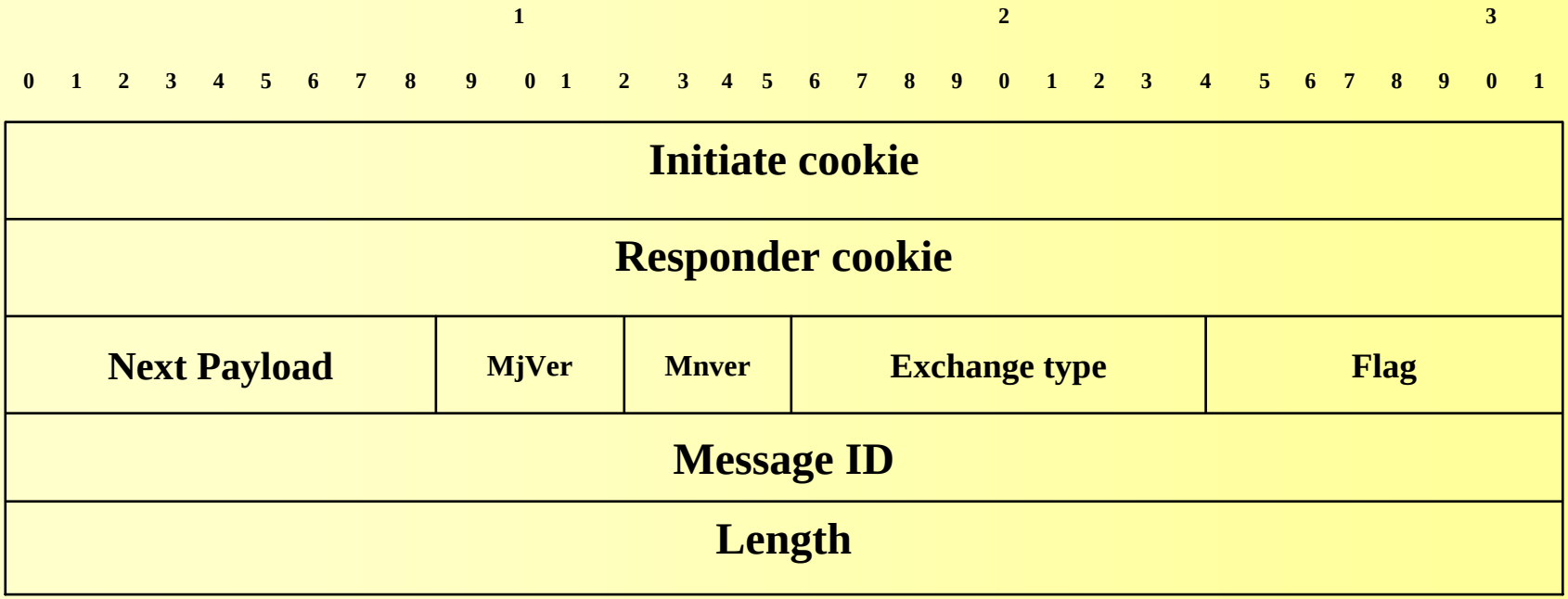
协商阶段

- 阶段1
  - 协商一个安全信道，为进一步协商具体的安全关联(阶段2)
- 阶段2
  - 为具体的安全协议协商安全关联
  - 可以协商多个SA

ISAKMP 载荷的类型

- **ISAKMP head**
- **Generic Payload Header**
- **Data Attributes**
- **Transform Payload**
- **Proposal Payload**
- **Security Association Payload**
- **Key Exchange Payload**
- **Identification Payload**
- **Certificate Payload**
- **Certificate Request Payload**
- **Hash Payload**
- **Signature Payload**
- **Nonce Payload**

ISAKMP 头

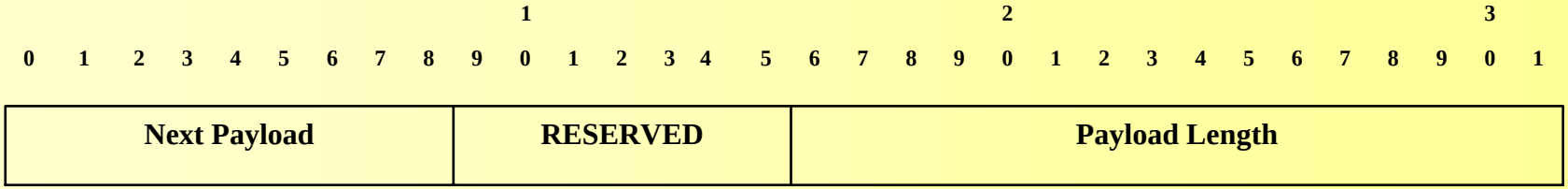

1
2
3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Initiate cookie
Responder cookie
Next Payload
MjVer
Mnver
Exchange type
Flag
Message ID
Length

交换类型

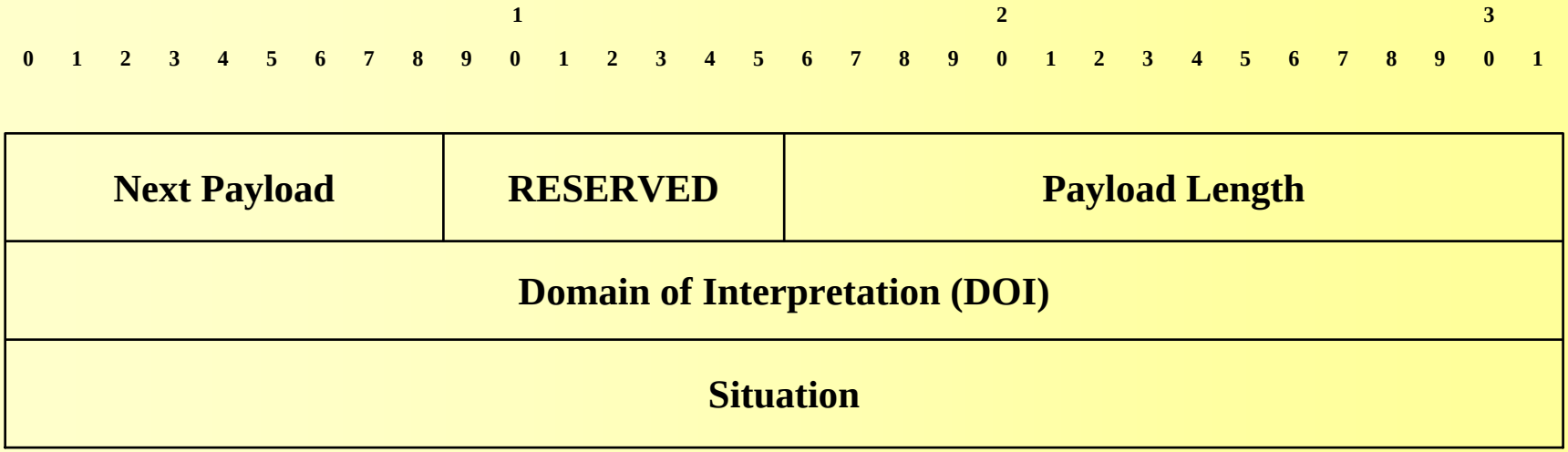
交换类型	值
NONE	0
Base	1
Identity Protection	2
Authentication Only	3
Aggressive	4
Informational	5
ISAKMP Future Use	6 – 31
DOI Specific Use	32 – 239
Private Use	240 – 255

ISAKMP 通用载荷头

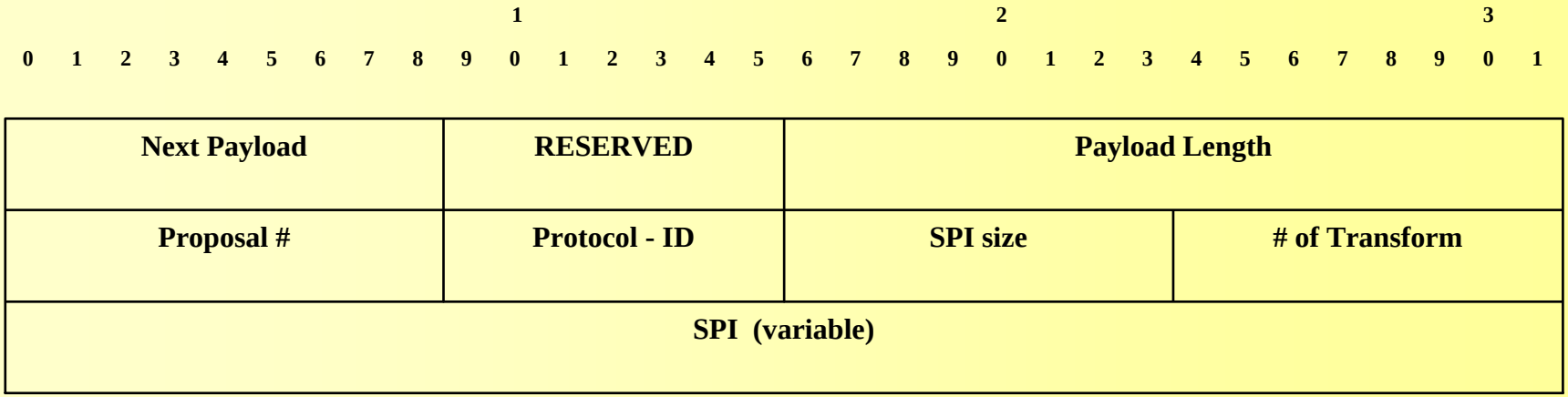
- 每一个ISAKMP载荷都以一个通用头开始，形成一个载荷的链，也为每个载荷定义了边界。



SA载荷


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 2 3 4 5 6 7 8 9 0 1
Next Payload
RESERVED
Payload Length
Domain of Interpretation (DOI)
Situation

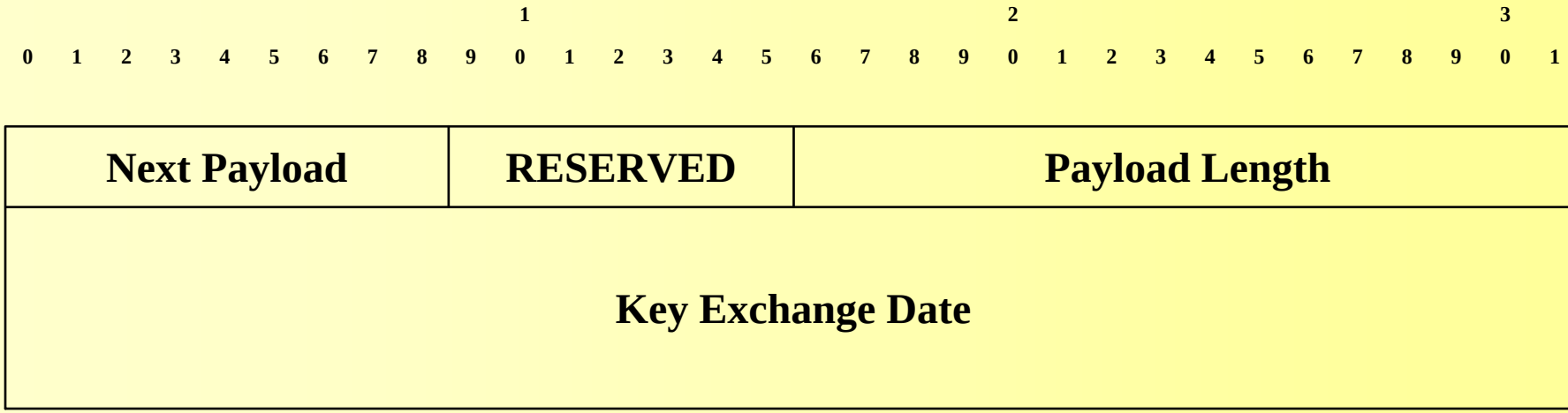
建议载荷


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
Next Payload
RESERVED
Payload Length
Proposal #
Protocol - ID
SPI size
# of Transform
SPI (variable)

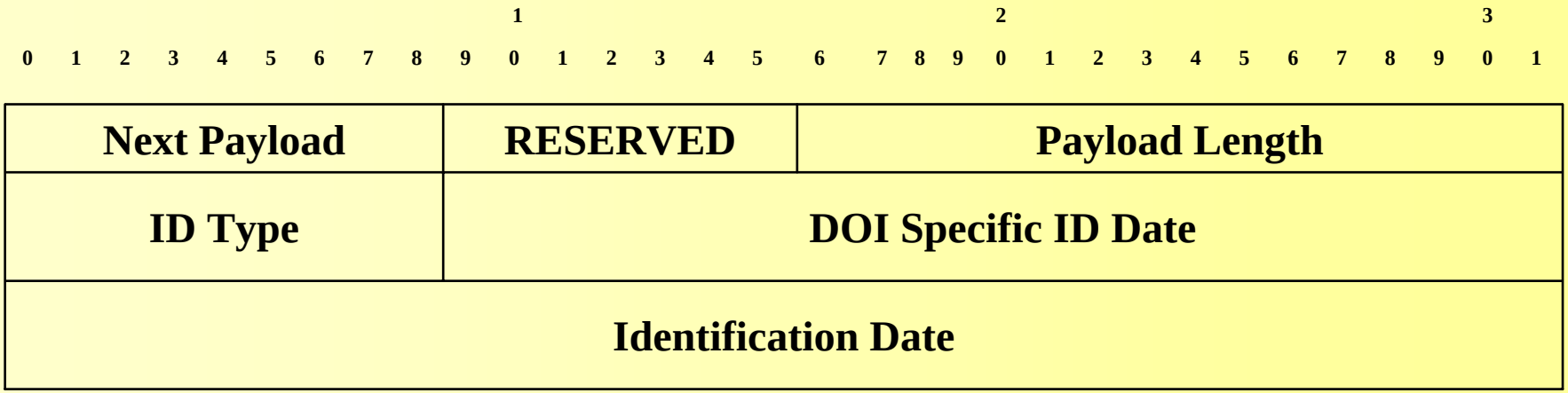
转码载荷

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
Next Payload	**RESERVED**	**Payload Length**	
Transform #	**Transform - ID**	**Reserved**	
SA attribute			

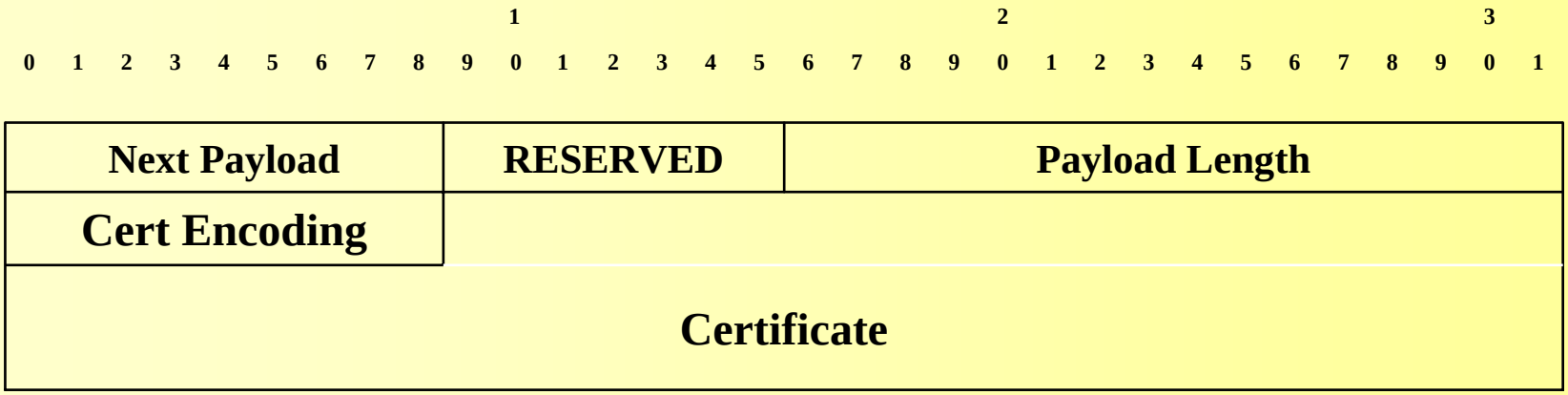
密钥交换载荷


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
Next Payload
RESERVED
Payload Length
Key Exchange Date

身份表示载荷


0 1 2 3 4 5 6 7 8 9 1 0 1 2 3 4 5 6 7 8 9 2 0 1 2 3 4 5 6 7 8 9 3 0 1
Next Payload
RESERVED
Payload Length
ID Type
DOI Specific ID Date
Identification Date

证书载荷


0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
Next Payload
RESERVED
Payload Length
Cert Encoding
Certificate

例子

P1: AH
- T1: HMAC-SHA
- T2: HMAC-MD5

P2: ESP
- T1: use HMAC-SHA with 3DES
- T2: use HMAC- MD5 with 3DES
- T3: use HMAC- SHA with DES
- T4: use HMAC-MD5 with DES

P3: ESP
- T1: use HMAC-SHA with 3DES
- T2: use HMAC- MD5 with 3DES
- T3: use HMAC-SHA with DES
- T4: use HMAC-MD5 with DES

P3: PCP
- T1: LZS
- T2: Deflate

例子的语义

(AH-HMAC-SHA OR AH-HMAC-MD5)

OR

(3DES HMAC-SHA OR 3DES HMAC-MD5 OR DES HMAC-SHA OR DES HMAC-MD5)

OR

[(3DES HMAC-SHA OR 3DES HMAC-MD5 OR DES HMAC-SHA OR DES HMAC-MD5)

AND (PCP-LZS OR PCP-DEFLATE)

]

7. The Internet Key Exchange(IKE)

- RFC2409，The Internet Key Exchange (IKE)，D. Harkins，D. Carrel.
- IKE的目标
  - 协商并验证一个密钥相关数据的交换;
  - 用于安全关联的建立。

SKEME

- Hugo Krawczyk, SKEME A Versatile Secure Key Exchange Mechanism for Internet, IEEE Proceedings of the Symposium on Network and Distributed Systems Security, 1996.

基本要求

- 保密性和验证
- 密钥的新鲜性
- 完善的前向保密性(Perfect Forward Secrecy (PFS))
- 保密性与匿名性
- 抗拥塞攻击

保密性和验证

- 只有交换的参与者才能获取交换的密钥相关的数据。
- 交换的数据必须是新鲜的和唯一的。

密钥的新鲜性

- 密钥交换协议必须提供定期更新密钥的机制
  - 需要频繁更新的数据采用耗费更低的机制;
  - 耗费更高的机制用于需要安全程度更高、频率更低的数据交换。

完善的前向保密性 PFS（1）

- 考虑在象密钥泄漏时使得损失尽可能地小。
- 使得在密钥泄漏之前的交换的密钥、加密的数据都仍然是安全的；损失仅限于密钥泄漏之后继续使用泄漏的密钥（没有发现密钥的泄漏）。

完善的前向保密性 PFS（2）

- 如果会话密钥的交换是通过用A的公钥加密的，则当A的私钥泄漏后，过去用A的公钥交换的会话密钥都可能泄漏（例如，攻击者在还没有得到A的私钥时就截取密钥交换的报文和加密数据的报文，等待获取A的私钥后处理）
- 如果用D-H方法交换密钥，交换是利用双方的公钥验证接收到的D-H公开数的完整性，就可以达到完善的前向保密性的目标。
  - 攻击者可以利用获得的A的私钥进行假冒A的身份（如果A还没有发现），但不能对获得过去交换的密钥。

设(k_{pB}, k_{sB}), (k_{pA}, k_{sA}) 分别是A和B的(公钥,私钥)对

	A		B
1	$M_1=E(k_{pB}, K_1)$	→	
2	$M_2=E(k_{pB}, K_2)$	→	
	……		
t	$k_1=E(k_{sB}, M_1)$, $k_2=E(k_{sB}, M_2)$		

	A		B
1	$X1=g^{x1} \bmod n$, $E(k_{sA}, X1)$	→	**$X1=?\ E(k_{pA}, E(k_{sA}, X1))$**
	$Y1=?\ E(k_{pB}, E(k_{sB}, Y1))$	←	$Y1=g^{y1} \bmod n$, $E(k_{sB}, Y1)$
2	$X2=g^{x2} \bmod n$, $E(k_{sA}, X2)$	→	
		←	$Y2=g^{y2} \bmod n$, $E(k_{sB}, Y2)$
	……		
t	**能从k_{sA},k_{sB}获得$g^{x1\cdot y1} \bmod n$, $g^{x2\cdot y2} \bmod n$吗?**		

保密性和匿名性

- 对通信的各方交换的数据保密
- 隐藏交换数据各方的身份
  - 通信方的IP地址等身份信息

防止拥塞攻击

- 完全阻止拥塞是很困难的。
- 一些预防性的措施可以减轻拥塞攻击的影响。
- 在协议中涉及到公开密钥算法操作的地方往往存在拥塞攻击的可能，因为公开密钥算法操作非常耗时。
- 利用简单的 cookies 技术使得这样的攻击更加困难
  - **P. Karn and W A Simpson, The Photuris, Session Key Management Protocol, Internet Draft draft-ietf-ipsec-photuris-03.txt, September, 1995.**
  - **RFC2522, Photuris: Session-Key Management Protocol，P. Karn，W. Simpson.**

基本协议和它的阶段（1）

- 共享数协商阶段
  - A→ B: PKE_B（K_A)
  - B → A: PKE_A（K_B)
  - K_0= H(K_A, K_B)

如果发起方的身份需要的保密的话，第一个消息也需要加密发起方的身份:

A→ B: PKE_B（id_A, K_A)

基本协议和它的阶段（2）

- 交换阶段
  - A→ B: $g^x \bmod p$
  - B → A: $g^y \bmod p$

- 认证阶段
  - A→ B: $F_{K0}(g^y, g^x, id_A, id_B)$
  - B → A: $F_{K0}(g^x, g^y, id_B, id_A)$

基本协议和它的阶段（3）

- 各个阶段的综合

  A→ B: $PKE_B(id_A, K_A)$， g^x ,

  B→ A: $PKE_A(K_B , id_B), g^y , F_{K0}(g^x, g^y, id_B , id_A)$

  A→ B: $F_{K0}(g^y, g^x, id_A , id_B)$

  $SK= H (g^{xy} \bmod p)$

SKEME — 无PFS特性

- 共享数交换
  - A→ B: PKEB（KA)
  - B → A: PKEA（KB)
  - K0= H(KA, KB)
- 交换
  - A→ B: nonceA
  - B → A: nonceB
- 认证
  - A→ B: $F_{K0}(nonce_B, nonce_A, id_A, id_B)$
  - B → A: $F_{K0}(nonce_A, nonce_B, id_B, id_A)$
- SK= $F_{K0}(arg)$, arg=$F_{K0}(nonce_B, nonce_A, id_A, id_B)$

协议的组合

$PKE_B(id_A, K_A)$, $value_A$ →

$PKE_A(id_A, K_A)$, $value_A$, $F_{K0}(valua_A, value_B, id_B, id_A)$ ←

$PKE_B(id_A, K_A)$, $value_A$, $F_{K0}(valua_B, value_A, id_A, id_B)$ →

模式	PK	DH	**$Value_A$**	**$Value_B$**	K_0	SK	组成
基本	Y	Y	g^x	g^y	**$H(K_A, K_B)$**	$H(g^{xy})$	PK+PFS
No PFS	Y	N	**$nonce_A$**	**$nonce_B$**	**$H(K_A, K_B)$**	**F_{K0}(arg）**	**高效无PFS**
预共享	N	Y	g^x	g^y	预共享	$H(g^{xy})$	PFS
快速	N	N	**$nonce_A$**	**$nonce_B$**	预共享	**F_{K0}(arg）**	

IKE 模式

- 第一阶段两种模式
  - 主模式(Main Mode)
  - 野蛮模式 (Aggressive Mode)

密钥材料的协商

- 预共享验证

  SKEYID = prf(pre-shared-key, Ni_b | Nr_b)

- 公开密钥验证

  SKEYID = prf(hash(Ni_b | Nr_b), CKY-I | CKY-R)

- 签名验证

  SKEYID = prf(Ni_b | Nr_b, g^xy)

- 密钥材料的计算
  - **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)**

IKE阶段1 预共享密钥验证方式

主模式

发起者	方向	响应者
HDR, SA	→	
	←	**HDR, SA**
HDR, KE, Ni	→	
	←	**HDR, KE, Nr**
HDR*, IDii, HASH_I	→	
	←	**HDR*, IDir, HASH_R**

野蛮模式

发起者	方向	响应者
HDR, SA, KE, Ni, IDii	→	
	←	**HDR, SA, KE, Nr, IDir, HASH_R**
HDR, HASH_I	→	

IKE阶段1 预共享密钥验证方式

- **SKEYID = prf(pre-shared-key, Ni_b | Nr_b)**
- **SKEYID_d = prf(SKEYID, $g^{xi \cdot xr}$ | CKY-I | CKY-R | 0)**
- **SKEYID_a = prf(SKEYID, SKEYID_d | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 1)**
- **SKEYID_e = prf(SKEYID, SKEYID_a | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 2)**

- 若用**main mode**交换，**SKEYID_e** 的计算在**IDi**交换之前，所以只能建立在对方的**IP**地址的基础上。
- 野蛮模式不能保护身份信息。
- **HASH_I, HASH_R** 用于对协商的**SA** 进行验证。
  - **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)**

主模式

发起者	方向	响应者
HDR, SA	→	
	←	**HDR, SA**
HDR, KE, [HASH(1),] **{ IDii_b }PubKey_r,** **{ Ni_b } PubKey_r**	→	
	←	**HDR, KE,** **{ IDir_b } PubKey_i,** **{ Nr_b } PubKey_i**
HDR*, HASH_I	→	
	←	**HDR*, HASH_R**

野蛮模式

发起者	方向	响应者
HDR, SA, [HASH(1),] KE, {IDii_b}Pubkey_r, {Ni_b}Pubkey_r	→	
	←	**HDR, SA, [HASH_R,] KE, {IDir_b}PubKey_i, {Nr_b}PubKey_i,**
HDR, HASH_I	→	

Aggressive 模式也能提供身份的保护。

IKE阶段1 公开密钥验证方式

- **SKEYID = prf(hash(Ni_b|Nr_b), CKY-I, CKY-R)**
- **SKEYID_d = prf(SKEYID, $g^{xi \cdot xr}$ | CKY-I | CKY-R | 0)**
- **SKEYID_a = prf(SKEYID, SKEYID_d | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 1)**
- **SKEYID_e = prf(SKEYID, SKEYID_a | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 2)**

- **HASH_I, HASH_R 用于对协商的SA 进行验证。**

  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)

- **HASH(1)** 是对发起者采用的证书的散列值，使得在响应者有多个公钥的情况下可以选择哪一把私钥解密消息。
- 身份和**Nonce**载荷分开加密是使得可以确定**Nonce**载荷的长度

IKE阶段1　改进的公开密钥验证方式

主模式

发起者	方向	响应者
HDR, SA	→	
	←	**HDR, SA**
HDR, [HASH(1),] {Ni_b}Pubkey_r, {KE_b}Ke_i, {IDii_b}Ke_i, [{Cert-I_b}Ke_i]	→	
	←	**HDR, {Nr_b}PubKey_i, {KE_b}Ke_r,{IDir_b}Ke_r**
HDR*, HASH_I	→	
	←	**HDR*, HASH_R**

Ne_i=prf(Ni_b, CKY_I)　→ Ke_i
Ne_r=prf(Nr_b, CKY_R)　→ Ke_r

IKE 阶段 1　签名验证方式

主模式

发起者	方向	响应者
HDR, SA	→	
	←	**HDR, SA**
HDR, KE, Ni	→	
	←	**HDR, KE, Nr**
HDR*, IDii, [CERT,] SIG_I	→	
	←	**HDR*, IDir, [CERT,] SIG_R**

野蛮模式

发起者	方向	响应者
HDR, SA, KE, Ni, IDii	→	
	←	**HDR, SA, KE, Nr, IDir, [CERT,] SIG_R**
HDR, [CERT,] SIG_I	→	

IKE 阶段 1 签名验证方式

- **SKEYID = prf(Ni_b | Nr_b, $g^{xi \cdot xr}$)**
- **SKEYID_d = prf(SKEYID, $g^{xi \cdot xr}$ | CKY-I | CKY-R | 0)**
- **SKEYID_a = prf(SKEYID, SKEYID_d | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 1)**
- **SKEYID_e = prf(SKEYID, SKEYID_a | $g^{xi \cdot xr}$ | CKY-I | CKY-R | 2)**

- **SIG_I和SIG_R是对 HASH_I 和HASH_R签名**
- 由于采用数字签名，可以防止抵赖。

第8章 安全套接字层

Security Socket Layer

参考文献

- The SSL protocol, version 3.0, Netscape Communications Corporation, March 1996.
- William Stallings, 密码编码学与网络安全-原理与实践，第14.2章。

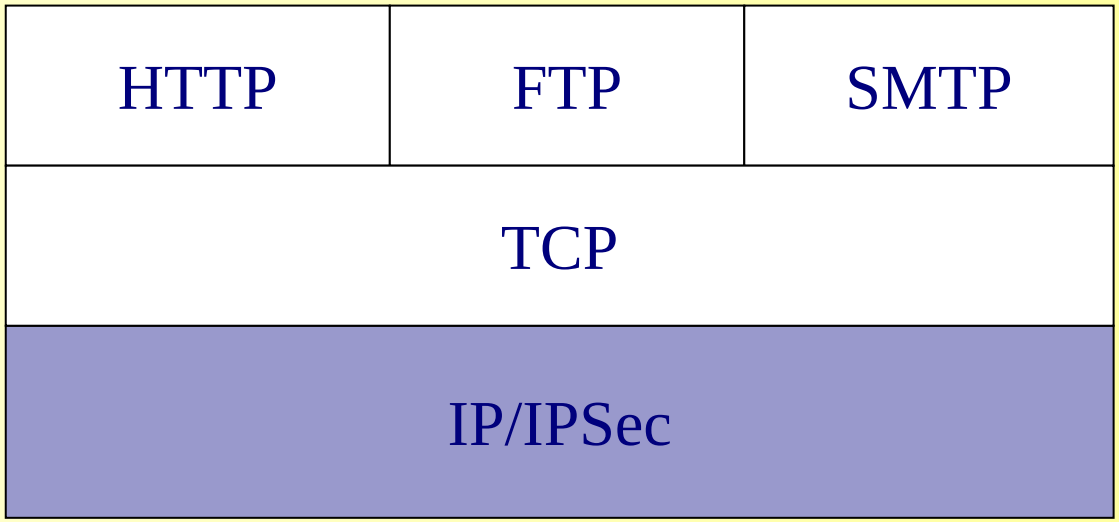
SSL的历史

- SSLv2最初在1995年用在Netscape公司的Netscape Navigator1.1中；
- Microsoft 公司在SSLv2之上做了些改进，修补了一些安全问题，提出了一种类似的协议PCT(Private Communication Technology)。
- 之后Netscape公司对该协议进行了彻底的改进，得到了SSLv3。
- IETF希望提出统一的协议，提出了类似SSLv3的协议TLS(Transport Layer Security)。

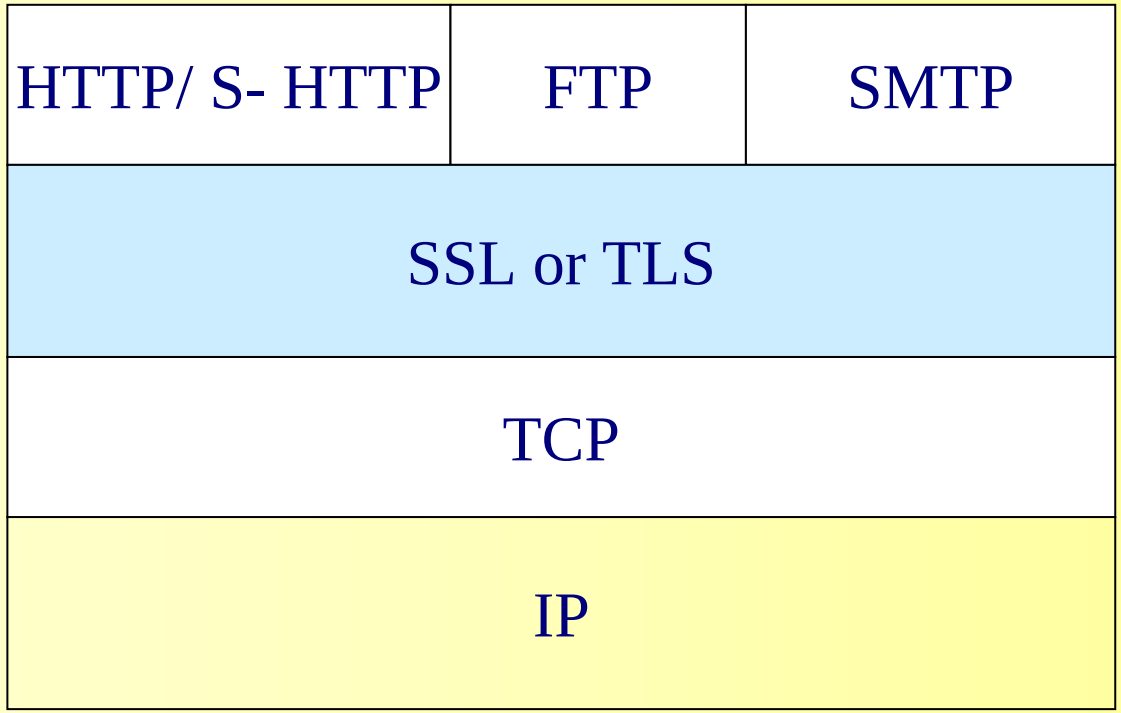
内容

- SSL概念
- SSL 记录层
- SSL 握手协议

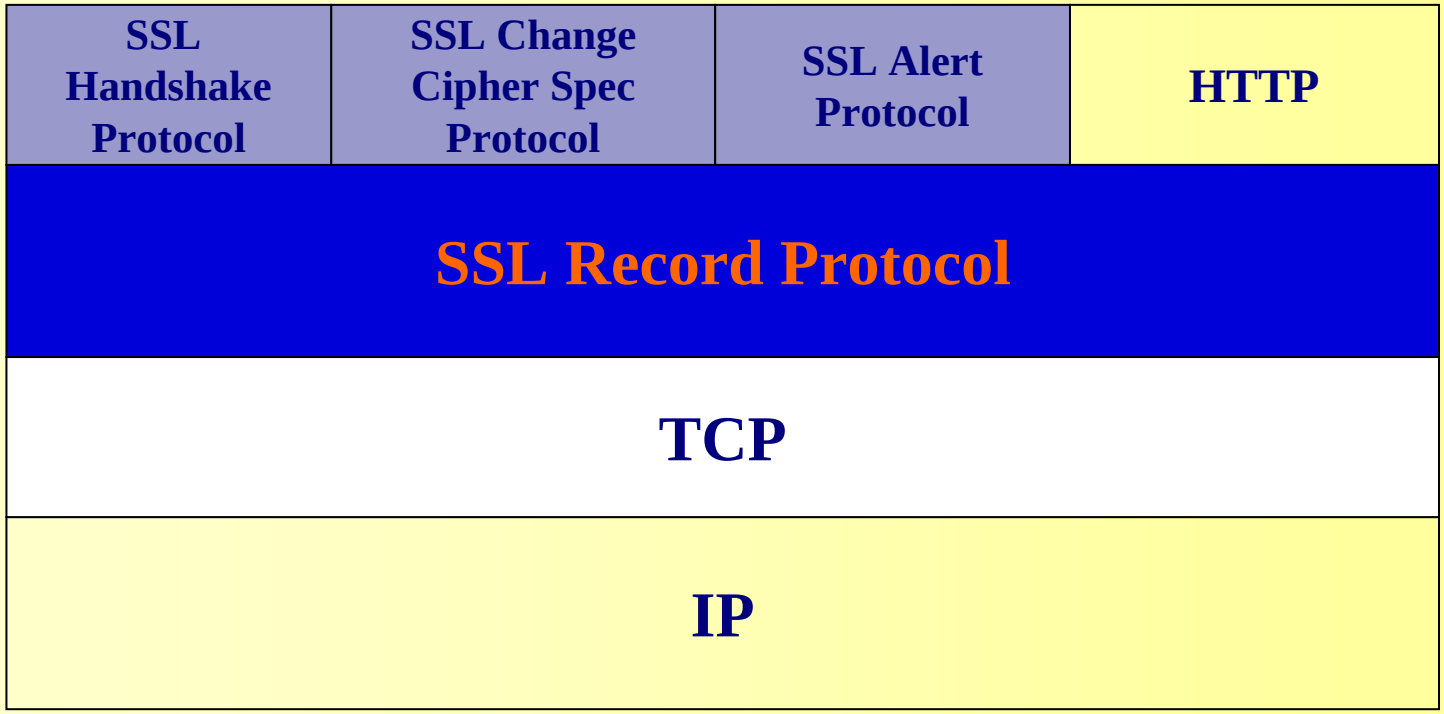
网络层安全



运输层安全



SSL 协议栈


SSL Handshake Protocol
SSL Change Cipher Spec Protocol
SSL Alert Protocol
HTTP
SSL Record Protocol
TCP
IP

SSL 的目标

- SSL的主要目标是在两个应用程序之间提供机密、可靠服务
- SSL协议包含两个协议层
- The protocol is composed of two layers.
  - 底层是SSL记录层： **SSL Record Protocol 。这个协议层主要用于为高层协议提供保密包装服务。**
  - SSL握手协议为服务器和客户端之间提供在收发数据之前协商加密算法、密钥的服务。
- 与应用协议无关
  - SSL对高层的应用协议透明.

SSL *连接安全*

- 通信数据是保密的.
- 通信的对方可以验证
- 通信连接是可靠的.
- 提供数据完整性服务

会话状态

- 会话标识
- 对方证书
- 压缩算法
- 密码参数
  - 加密算法 、 消息鉴别码算法、摘要值长度
- 主密钥
- 可重用性

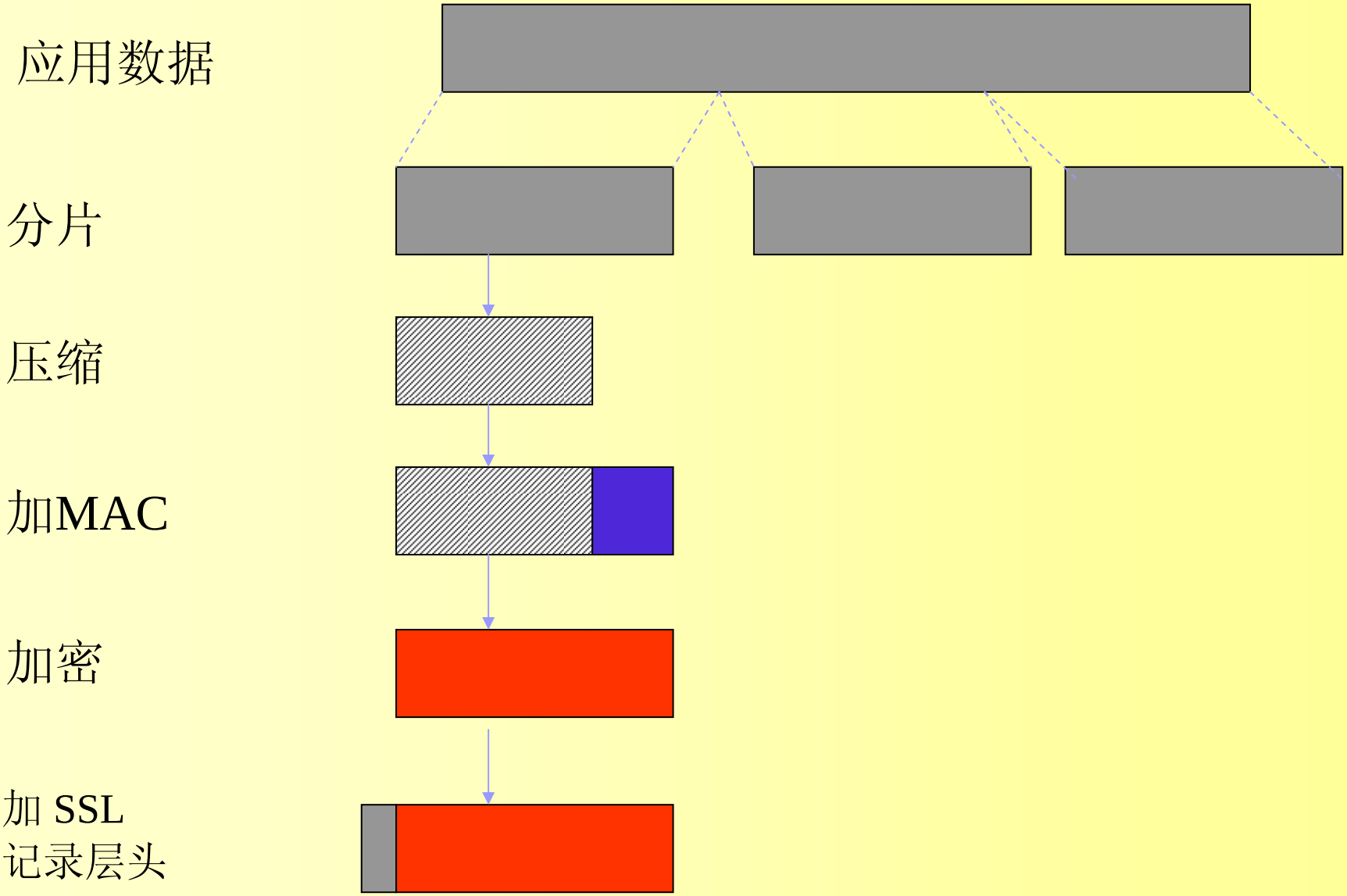
连接状态

- 服务器和客户端随机数
- 服务器写MAC密钥
- 客户端写MAC密钥
- 服务器写密钥
- 客户端写密钥
- 初始化向量
- 序列号

SSL 记录层

- SSL Record Protocol为SSL连接提供两种服务
  - 保密性。利用Handshake Protocol定义一个共享的保密密钥用于对SSL有效负载加密。
  - 消息完整性。利用Handshake Protocol定义一个共享的MAC写密钥用于形成MAC。

SSL记录层



SSL 记录层 — 支持的上层协议

- 改变密码规格消息 (ChangeCipherSpec)
- 报警协议 (Alert protocol)
- 握手协议 (Handshake protcol)
- 应用协议

CipherSpec

CipherSpec 由Hand Shake协议协商确定，Record Layer协议使用。

```
struct {
    BulkCipherAlgorithm bulk_cipher_algorithm;
            // null, rc4, rc2, des, 3des, des40, fortezza
    MACAlgorithm mac_algorithm; // null, md5, sha
    CipherType cipher_type;          //  stream, block
    IsExportable   is_exportable     // true, false
    uint8    hash_size;
    uint8    key_material;
    uint8    IV_size;
} CipherSpec;
```

改变密码规格消息 (ChangeCipherSpec)

- 使连接的挂起状态转变为当前状态。
- 改变连接将要使用的密文族。

报警协议 (Alert protocol)

AlertDescription

close_notify	0
unexpected_message	10
bad_record_mac	20
decompression_failure	30
handshake_failure	40
no_certificate	41
bad_certificate	42
unsupported_certificate	43
certificate_revoked	44
certificate_expired	45
certificate_unknown	46
illegal_parameter	47

握手协议 (Handshake protcol)

- 运行在SSL 记录层之上
- 生成一个会话的密码参数
  - 协商协议的版本号
  - 协商加密算法
  - 认证
  - 协商共享密钥

协商一个新的会话

客户端		服务器
ClientHello	→	
	←	ServerHello
		Certificate*
		CertificateRequest*
		ServerKeyExchange*
Certificate*	→	
ClientKeyExchange		
CertificateVerify*		
change cipher spec		
Finished		
	←	change cipher spec
		Finished
Application Data	→	
	←	Application Data

更新一个旧的会话

客户端		服务器
ClientHello	→	
	←	**ServerHello**
		change cipher spec
		Finished
change cipher spec	→	
Finished		
Application Data	→	
	←	**Application Data**

握手协议支持的消息类型

hello_request
client_hello
server_hello
certificate
server_key_exchange
certificate_request
server_hello_done
certificate_verify
client_key_exchange
finished

ClientHello

```
struct {
    ProtocolVersion client_version;
        // 客户能理解的最高SSL版本
    Random random;
        // 32bit 的时间戳,28字节随机数，防止重放攻击。
    SessionID session_id;
        //  0:  在新的会话上创建新的连接
        //  非 0: 修改已存在的连接的参数
        //  或在这个会话上创建新的连接
    CipherSuite cipher_suites<2..2 16 -1>;
        // 客户支持的加密算法族
    CompressionMethod   compression_methods<1..2 8 -1>;
        // 客户支持的压缩算法族
} ClientHello;
```

CipherSuits

```
SSL_RSA_WITH_NULL_MD5                    = { 0x00,0x01 };
SSL_RSA_WITH_NULL_SHA                    = { 0x00,0x02 };
SSL_RSA_EXPORT_WITH_RC4_40_MD5           = { 0x00,0x03 };
SSL_RSA_WITH_RC4_128_MD5                 = { 0x00,0x04 };
SSL_RSA_WITH_RC4_128_SHA                 = { 0x00,0x05 };
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5       = { 0x00,0x06 };
SSL_RSA_WITH_IDEA_CBC_SHA                = { 0x00,0x07 };
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA        = { 0x00,0x08 };
SSL_RSA_WITH_DES_CBC_SHA                 = { 0x00,0x09 };
SSL_RSA_WITH_3DES_EDE_CBC_SHA            = { 0x00,0x0A };
SSS_DH_RSA_WITH_DES_CBC_SHA              = {           }
```

认证算法_加密算法_**MAC**算法

CipherSuite	IsExport-table	Key Exchange	Cipher	Hash
SSL_NULL_WITH_NULL_NULL	*	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	*	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	*	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	*	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5		RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA		RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	*	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA		RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	*	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA		RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA		RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA		DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA		DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA		DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA		DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DHE_DSS_EXPORT	DES40_CBC	SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA		DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA		DHE_DSS	3DES_EDE_CBC	SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DHE_RSA_EXPORT	DES40_CBC	SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA		DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA		DHE_RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	*	DH_anon_EXPORT	RC4_40	MD5
SSL_DH_anon_WITH_RC4_128_MD5		DH_anon	RC4_128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA		DH_anon	DES40_CBC	SHA
SSL_DH_anon_WITH_DES_CBC_SHA		DH_anon	DES_CBC	SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA		DH_anon	3DES_EDE_CBC	SHA
SSL_FORTEZZA_DMS_WITH_NULL_SHA		FORTEZZA_DMS	NULL	SHA
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA		FORTEZZA_DMS	FORTEZZA_CBC	SHA

ServerHello

```
struct {
    ProtocolVersion  server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```

ServerHello

- 版本字段包含了客户建议的最低版本和服务器支持的最高版本。
- 随机数字段是服务器生成的，独立于客户的随机数字段。
- 如果客户的会话ID非零， 服务器察看会话cache ，如果能够匹配并且服务器愿意利用已有的会话状态建立一个新的联接，则服务器返回同样的值；否则，服务器的会话ID字段包含了一个新会话的值。
- 密文族字段包含了服务器从客户建议密文族列表中选择的一个密文族。压缩字段包含了服务器从客户建议压缩列表中选择的一个压缩方法。

Certificate

- If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the **server hello** message.

- The certificate is a sequence (chain) of X.509.v3 certificates, ordered with the sender's certificate first and the root certificate authority last.

三种认证模式

- 双向认证
- 服务器认证
- 匿名

匿名

- 匿名RSA
  - 客户端从Server Key Exchange消息获得服务器的公开密钥。
  - 客户端利用服务器公开密钥加密一个pre_master_secret。
  - 在 client key exchange 消息中传输。
  - 匿名RSA可能受到中间人攻击，中间人可能冒充服务器发送公开密钥。
  - pre_master_secret是保密的。

客户端		服务器
ClientHello	→	
	←	**ServerHello** **ServerKeyExchange***
ClientKeyExchange **change cipher spec** **Finished**	→	
	←	**change cipher spec** **Finished**
Application Data	→	
	←	**Application Data**

匿名RSA

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;

struct {
     opaque rsa_modulus<1..2^16-1>;
     opaque rsa_exponent<1..2^16-1>;
} ServerRSAParams;
```

- 匿名 Diffie-Hellman
  - 服务器的公开数在**server key exchange** 消息中传输
  - 客户端的公开数在**client key exchange** 消息中传输

客户端		服务器
ClientHello	→	
	←	**ServerHello** **ServerKeyExchange***
ClientKeyExchange **change cipher spec** **Finished**	→	
	←	**change cipher spec** **Finished**
Application Data	→	
	←	**Application Data**

匿名 Diffie-Hellman

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;

struct {
     opaque dh_p<1..2^16-1>;
     opaque dh_g<1..2^16-1>;
     opaque dh_Ys<1..2^16-1>;
} ServerDHParams;
```

RSA 密钥交换和认证

- 公开密钥可能包含在
  - 服务器的证书中
  - **server key exchange** 消息中的临时公钥
- 如果用RSA临时公开密钥来交换会话密钥，则用服务器Certificate中的公钥相应的私钥进行签名。
- 客户端用这个RSA临时公开密钥来加密pre_master_secret.
- 服务器解密得到pre_master_secret，生成finished 消息

RSA 密钥交换和认证

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;
struct {
     opaque rsa_modulus<1..216-
     1>;
     opaque rsa_exponent<1..216-
     1>;
} ServerRSAParams;
```

RSA 密钥交换和认证

客户端		服务器
ClientHello	→	
	←	ServerHello Certificate CertificateRequest* ServerKeyExchange*
Certificate* ClientKeyExchange change cipher spec Finished	→	
	←	change cipher spec Finished
Application Data	→	
	←	Application Data

Diffie-Hellman 密钥交换与认证

- D-H公开数的传递
  - 在证书中提供D-H公开数
  - 在**client key exchange** 消息和**server key exchange message** 中提供D-H公开消息，用证书签名；
- 公开数用hello.random连接后计算散列，防止重放攻击。

master_secret

master_secret =

MD5(pre_master_secret +

SHA(`A' + pre_master_secret +

ClientHello.random+ ServerHello.random))

+MD5(pre_master_secret +

SHA(`BB' + pre_master_secret + ClientHello.random +

ServerHello.random))

+ MD5(pre_master_secret +

SHA(`CCC' + pre_master_secret +ClientHello.random +

ServerHello.random));

key_block

key_block =

MD5(master_secret + SHA(`A' + master_secret +
ServerHello.random + ClientHello.random))
+MD5(master_secret + SHA(`BB' + master_secret +
ServerHello.random + ClientHello.random))
+MD5(master_secret + SHA(`CCC' + master_secret +
ServerHello.random +ClientHello.random))
+ [...];