Linux IPsec

--IKEv2

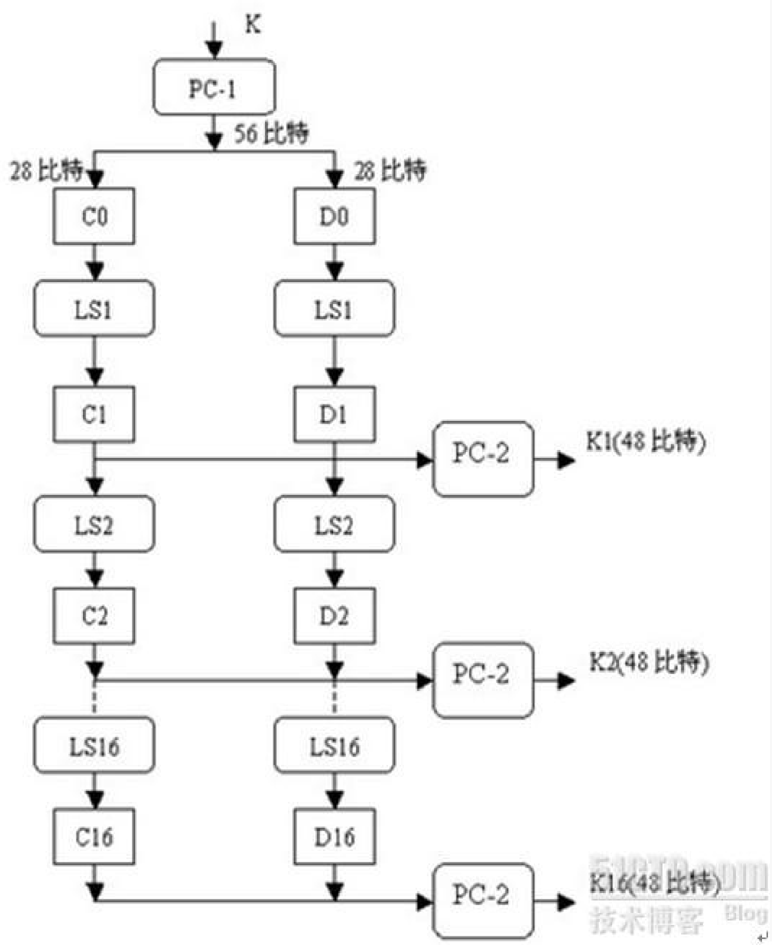
VPN

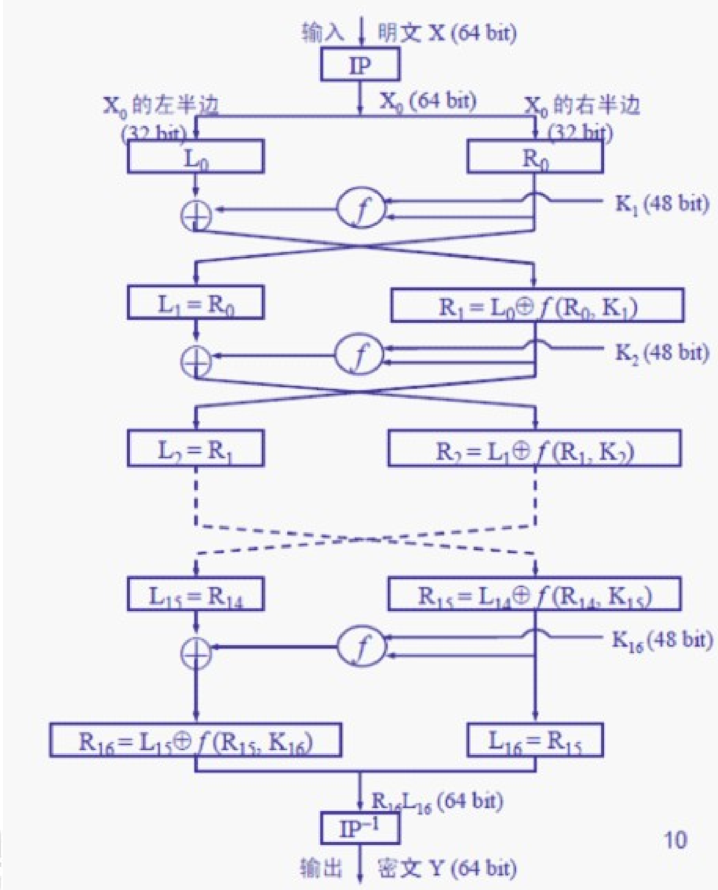
- SSL VPN (OPENVPN)
- IPSEC VPN (LIBRISWAN/FREESWAN/STRONGSWAN/IPSEC-TOOLS)
- L2TP/PPTP VPN (XL2TPD)

IPSEC

- Security Protocol
- Algorithm
- IKE

DES(64, 56) --confidentiality





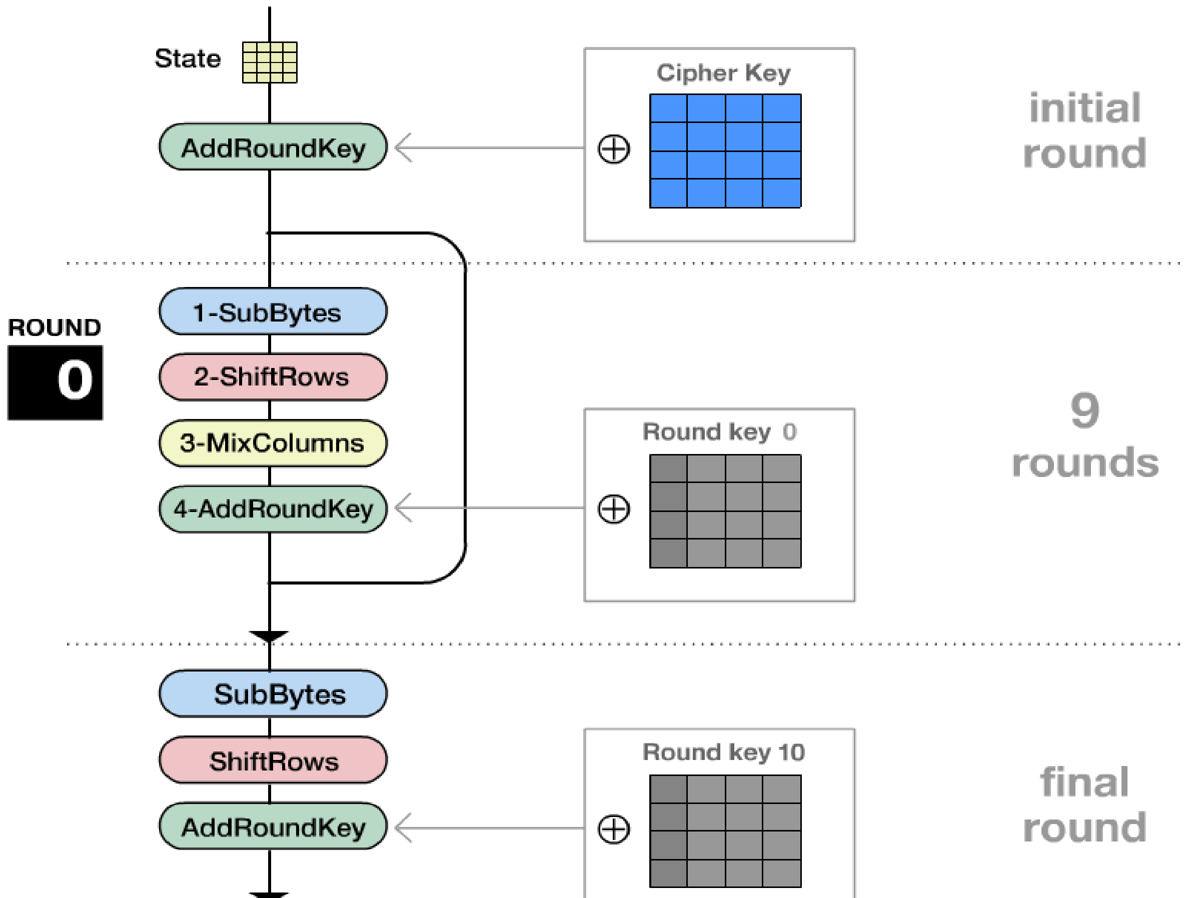
3DES --confidentiality

Ciphertext = EK3(DK2(EK1(Plaintext)))

Plaintext = DK1(EK2(DK3(Ciphertext)))

- Electronic codebook(ECB)
- Cipher-block chaining(CBC)
- Cipher feedback(CFB)
- output feedback(OFB)
- Counter mode(CTR)

AES(128, 128/192/256) --confidentiality



RSA

Key Generation	
Select p, q	p, q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

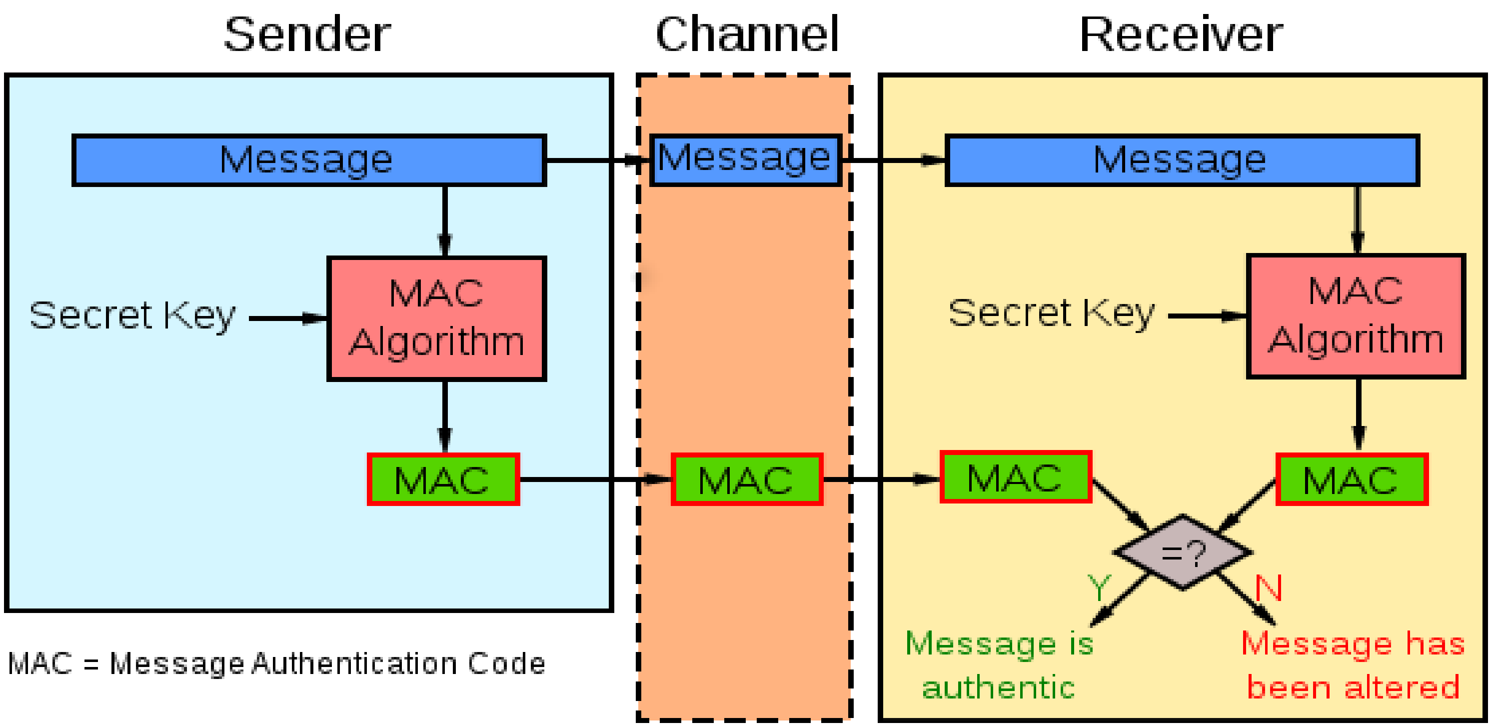
Appendix B: RSA Proof using Fermat's little theorem

SHA --integrity

- SHA-0
- SHA-1(160) base of md4
- SHA-2(SHA-256/384/512)
- SHA-3(224/256/384/512), sha-1/md5 is attracked

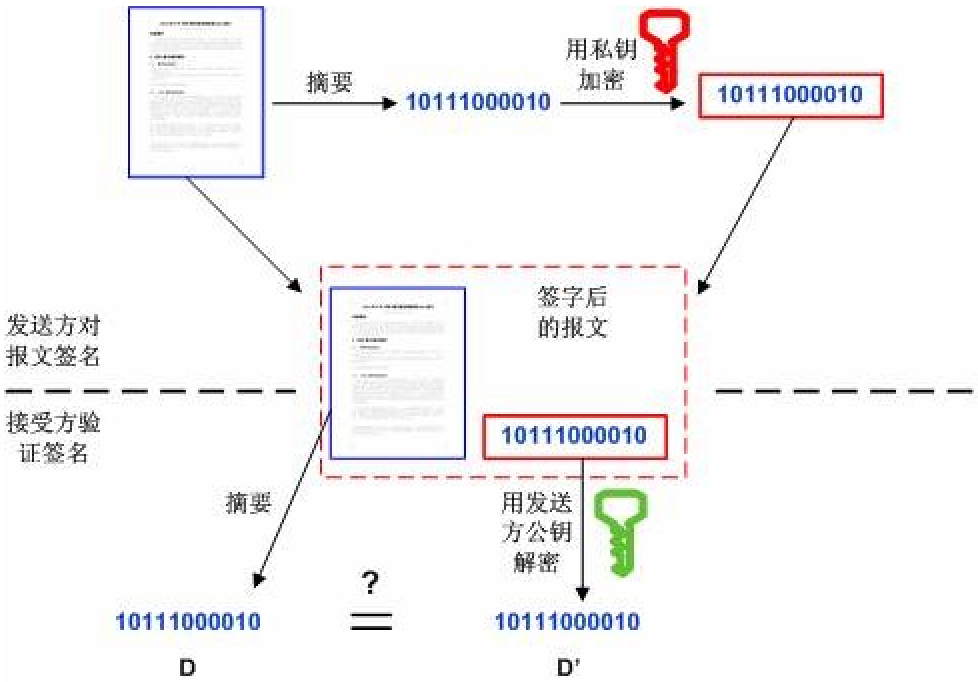
MAC --integrity

- HMAC
- CMAC

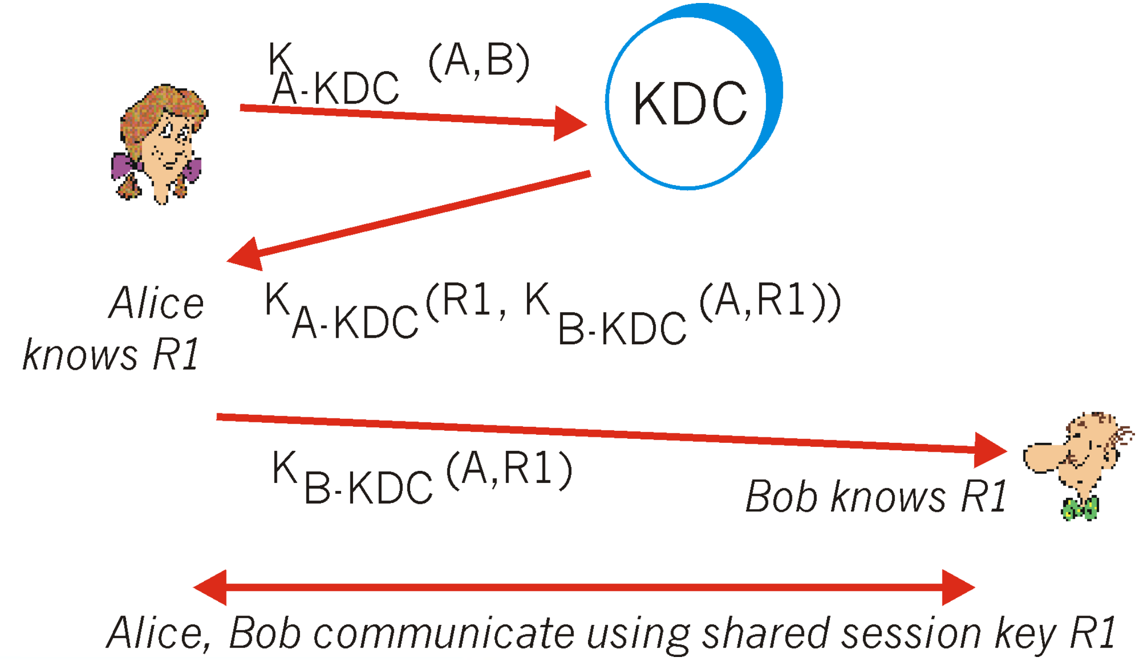


Digital Signature –integrity/ Recipient denied

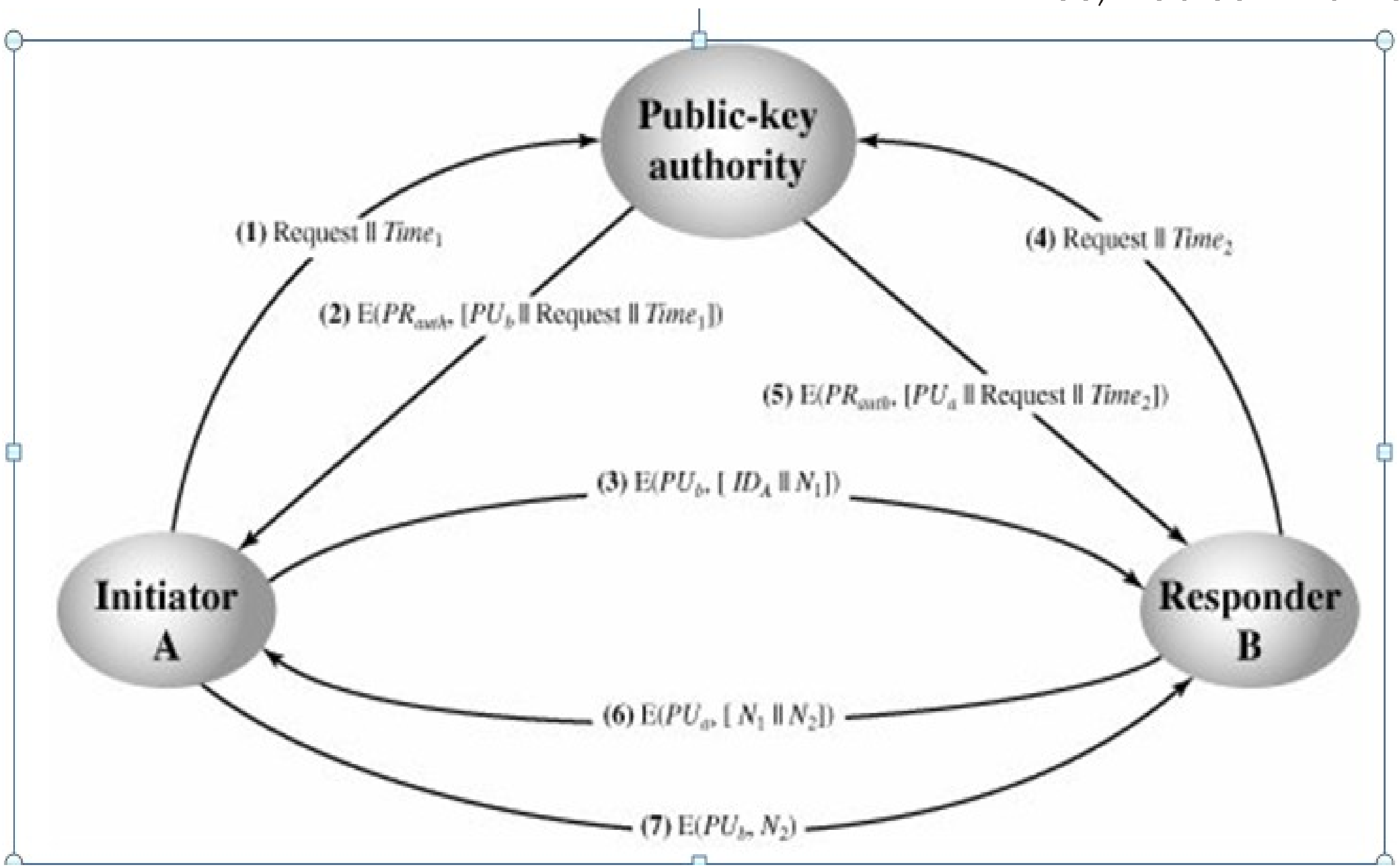
Use SHA



Certificate (x.509) --mutual trust

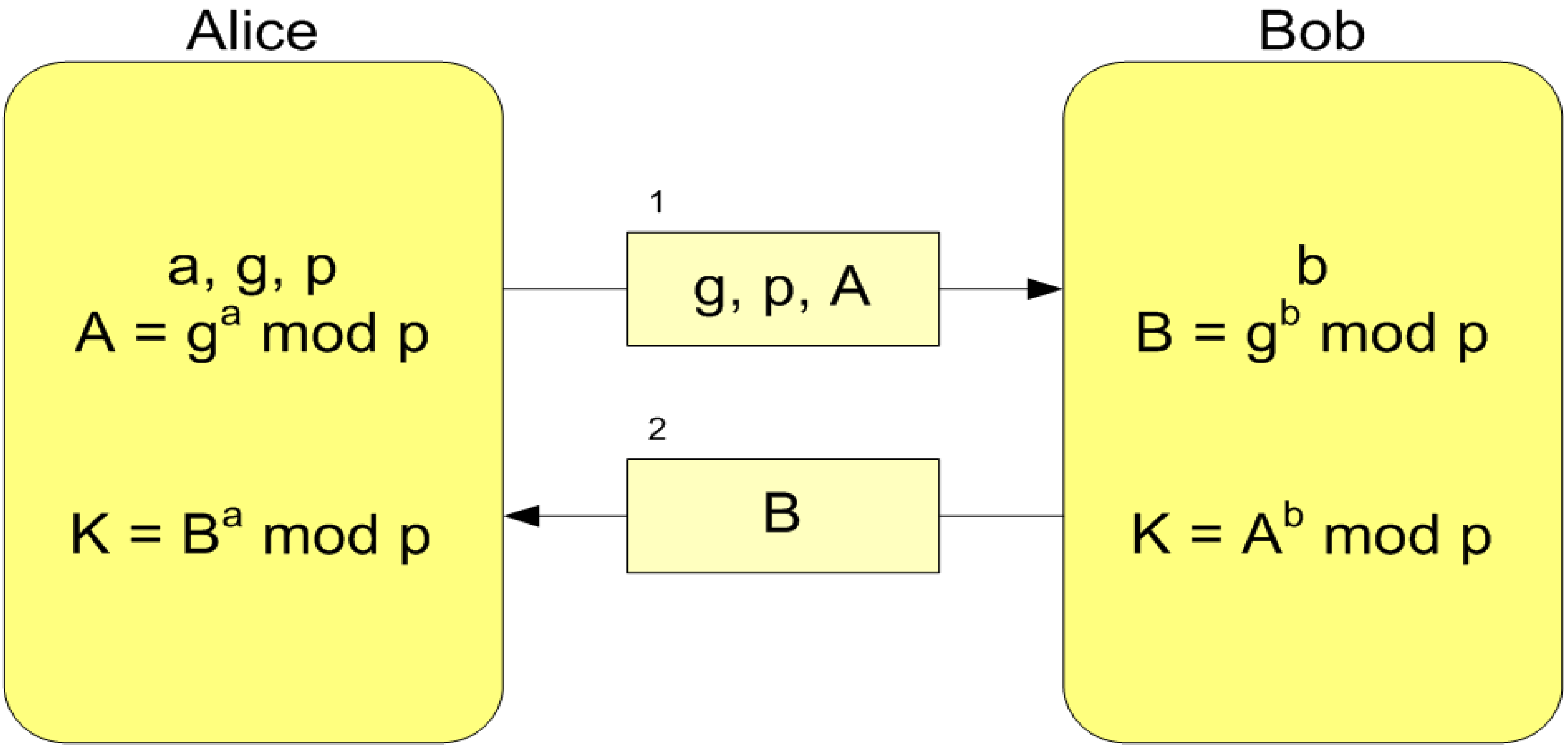


asymmetric encryption mutual -way authentication



Appendix A:<u>create local ca and client/server certs</u>

Diffie-Hellman



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

discrete logarithm problem

Kerberos --mutual trust

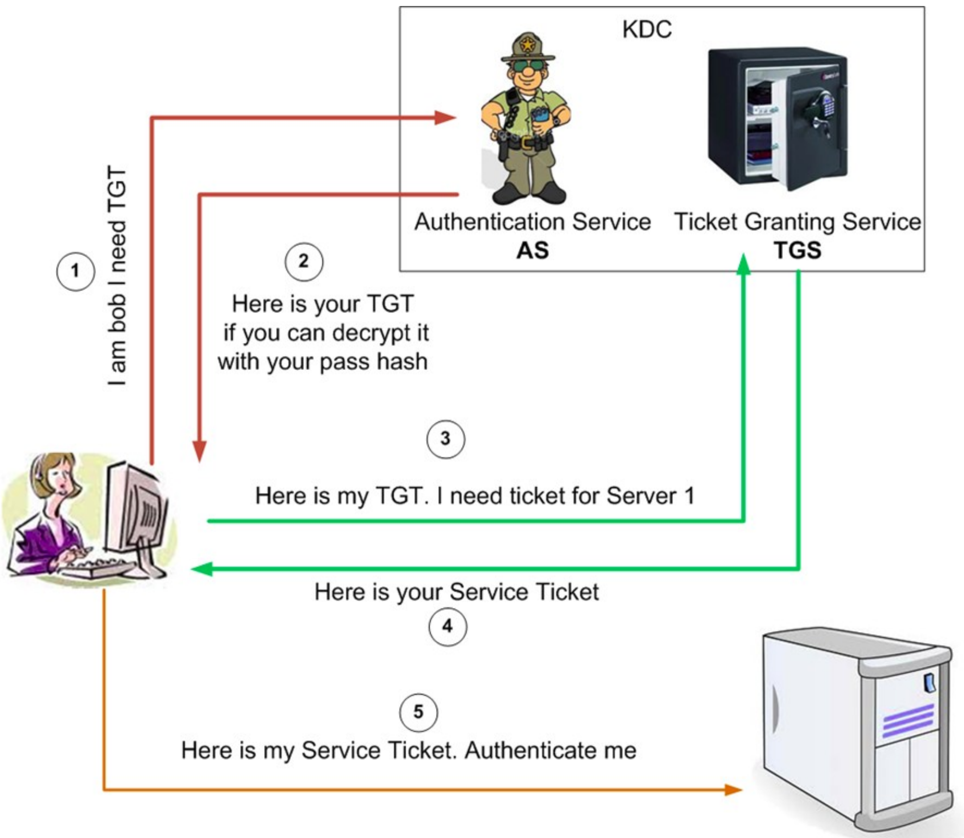


Table 15.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

<u>symmetric encryption mutual-way authentication</u>

IKEV2 --INITIATOR

- HDR:ISAKMP header
- SA: SA payload
- KE: key exchage payload
- AUTH: authentication payload
- ID: identification payload
- NONCE: incude Ni and Nr
- CERT: certificate payload
- CERTREQ: certificate request
- TS: traffic selection
- SK{...}: the encryption payload

1. IKE_SA_INIT i:

Start to Negotiate IKE SA

- HDR (SPI)
- SAi1
  - encryption algorithms
  - PRF algorithms
  - integrity algorithms
  - DH Groups (PFS)
- Ni, Anti-replay attack

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump1

2. IKE_SA_INIT r:

- HDR
- KEr:DH public
- Nr
- SAr
- (CERTREQ)

<u>Generate keys:</u>

- sk_d: derive new key for child sa
- sk_ai/ar:intregrity protection
- sk_ei/er:encrypting
- sk_pi/pr:generate auth payload

prf+ (K,S) = T1 | T2 | T3 | T4 | ...

where:

T1 = prf (K, S | 0x01)

T2 = prf (K, T1 | S | 0x02)

T3 = prf (K, T2 | S | 0x03)

T4 = prf (K, T3 | S | 0x04)

...

SKEYSEED = prf(Ni | Nr, **g^ir**)

{SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr }

= prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr)

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump2

3. IKE_AUTH i:

The same way to get keys

Start to Negotiate child SA

TS suppose the ip/port

Generate the Signature data

- HDR
- SK{IDi, Idr, AUTH, SAi2,TSi,TSr,(CERT, CERTREQ)}

MACedIDForI =**prf**(**SK_pi**, **IDType** | RESERVED | **InitIDData**

AUTH_DATA = prf(SK_pi, RealMessage1 | NonceRData |MACedIDForI)

Pre-shared key:

AUTH_DATA = prf(prf(Shared Secret, "Key Pad forIKEv2"), <InitiatorSignedOctets>

EAP:

When use key-generation:

AUTH_DATA = prf(prf(MSK(Master Shared Key), "Key Pad forIKEv2"), <InitiatorSignedOctets

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump3

4. IKE_AUTH r:

Decrypting data

the same way to generate the Signature data

generate the child sa's key meterails

- HDR
- SK{IDr, Idr, AUTH, SAr2,TSi,TSr,(CERT)}

For this:

KEYMAT = prf+(SK_d, Ni | Nr)

For create_child_sa:

KEYMAT = prf+(SK_d, g^ir (new) | Ni | Nr)

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump4

5. auth over

the same way to generate the child sa's key meterails

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump4

IKEV2 --CREATE_CHILD_SA

Initiator Responder

HDR, SK {SA, Ni, [KEi],

TSi, TSr} -->

<-- HDR, SK {SA, Nr, [KEr],

TSi, TSr}

- **Kei is something like DH pubic value**

IKEV2 --REKEYING

- Rekeying child_sa:

Initiator Responder

HDR, SK {N(REKEY_SA), SA, Ni, [KEi],

TSi, TSr} -->

<-- HDR, SK {SA, Nr, [KEr],

TSi, TSr}

- Rekeying ike_sa:

Initiator Responder

HDR, SK {SA, Ni, KEi} -->

<-- HDR, SK {SA, Nr, KEr}

- **child_sa tsi/tsr is different from the old**
- **ike_sa manage and control all the child_sa**

IKEV2 --NAT-T

- AH cannot go through Nat
- ESP
- IKE

request --> [N(COOKIE)],

SA, KE, Ni,

[N(NAT_DETECTION_SOURCE_IP)+,

N(NAT_DETECTION_DESTINATION_IP)],

[V+][N+]

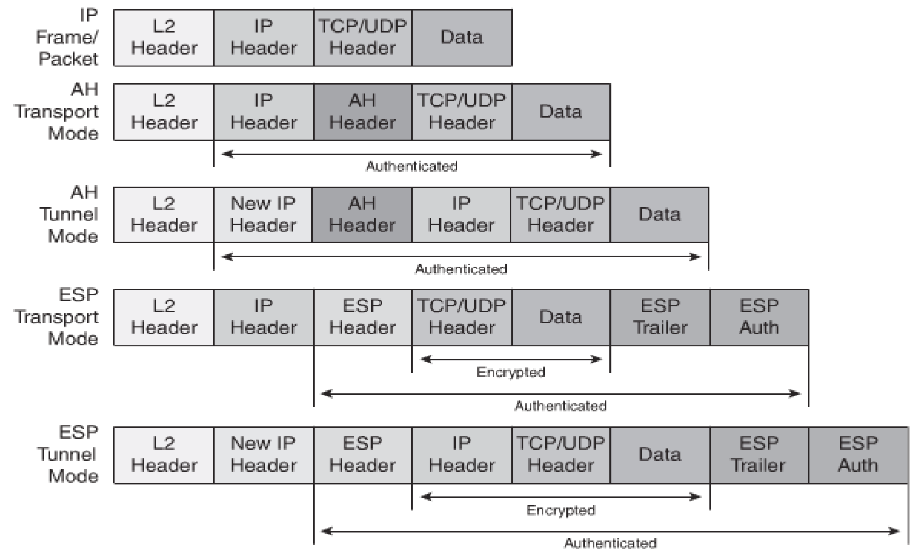
normal response <-- SA, KE, Nr,

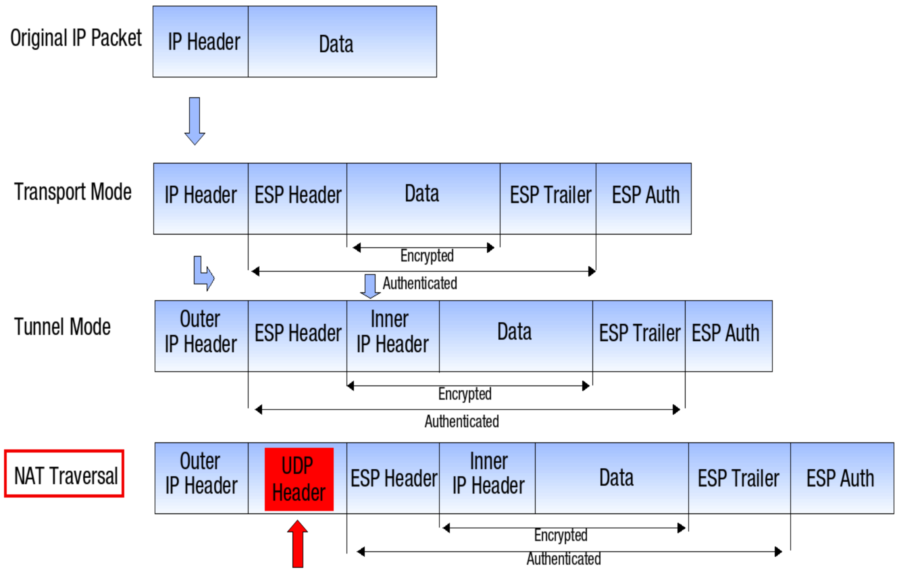
(no cookie) [N(NAT_DETECTION_SOURCE_IP),

N(NAT_DETECTION_DESTINATION_IP)],

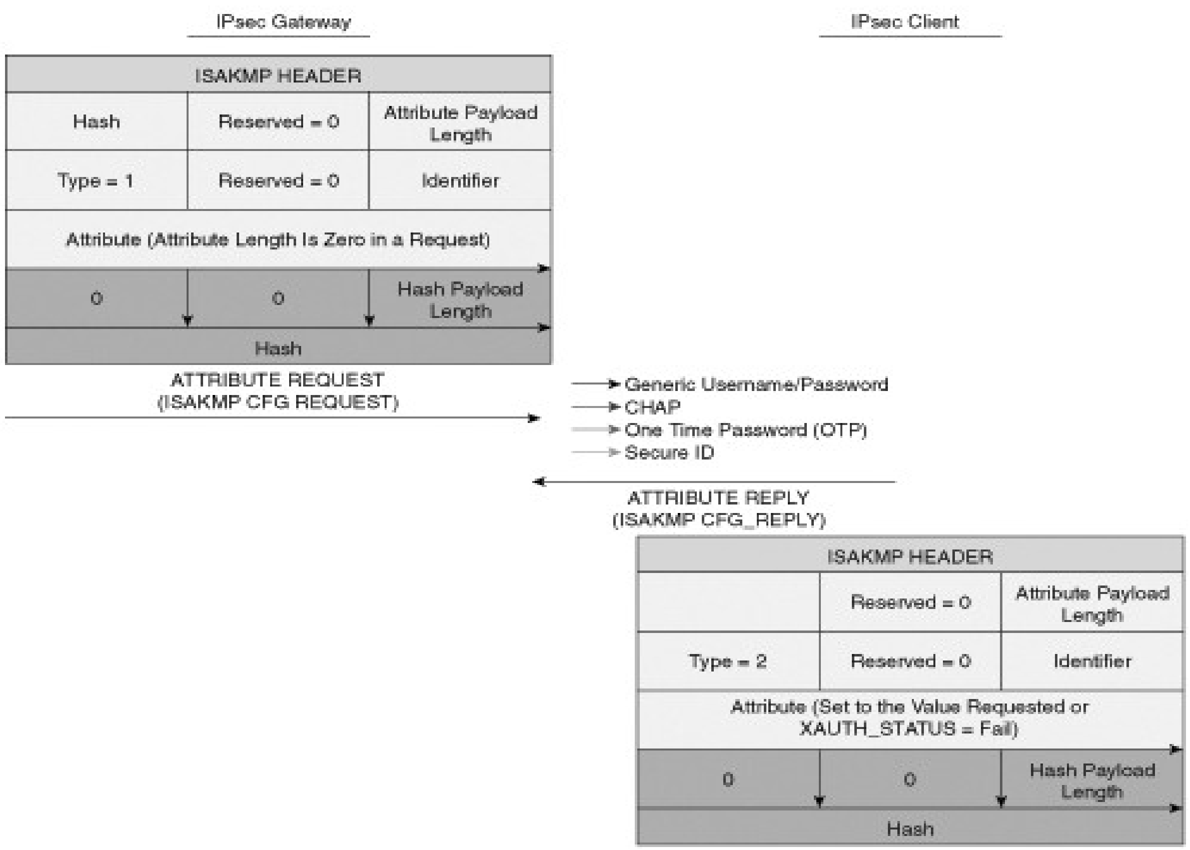
[[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],

[V+][N+]





IKEV2 --XAUTH


IPsec Gateway
IPsec Client
ISAKMP HEADER
Hash
Reserved = 0
Attribute Payload Length
Type = 1
Reserved = 0
Identifier
Attribute (Attribute Length Is Zero in a Request)
0
0
Hash Payload Length
Hash
ATTRIBUTE REQUEST
(ISAKMP CFG REQUEST)
Generic Username/Password
CHAP
One Time Password (OTP)
Secure ID
ATTRIBUTE REPLY
(ISAKMP CFG_REPLY)
ISAKMP HEADER
Reserved = 0
Attribute Payload Length
Type = 2
Reserved = 0
Identifier
Attribute (Set to the Value Requested or XAUTH_STATUS = Fail)
0
0
Hash Payload Length
Hash

IKEV2 --EAP

Initiator Responder

HDR, SAi1, KEi, Ni -->

<-- HDR, SAr1, KEr, Nr, [CERTREQ]

HDR, SK {IDi, [CERTREQ,]

[IDr,] SAi2,

TSi, TSr} -->

<-- HDR, SK {IDr, [CERT,] AUTH,

EAP }

HDR, SK {EAP} -->

<-- HDR, SK {EAP (success)}

HDR, SK {AUTH} -->

<-- HDR, SK {AUTH, SAr2, TSi, TSr }

IKEV2 --REQUEST ADDRESS

For example, message from initiator to responder:

CP(CFG_REQUEST)=

INTERNAL_ADDRESS()

TSi = (0, 0-65535,0.0.0.0-255.255.255.255)

TSr = (0, 0-65535,0.0.0.0-255.255.255.255)

NOTE: Traffic Selectors contain (protocol, port range, address

range).

Message from responder to initiator:

CP(CFG_REPLY)=

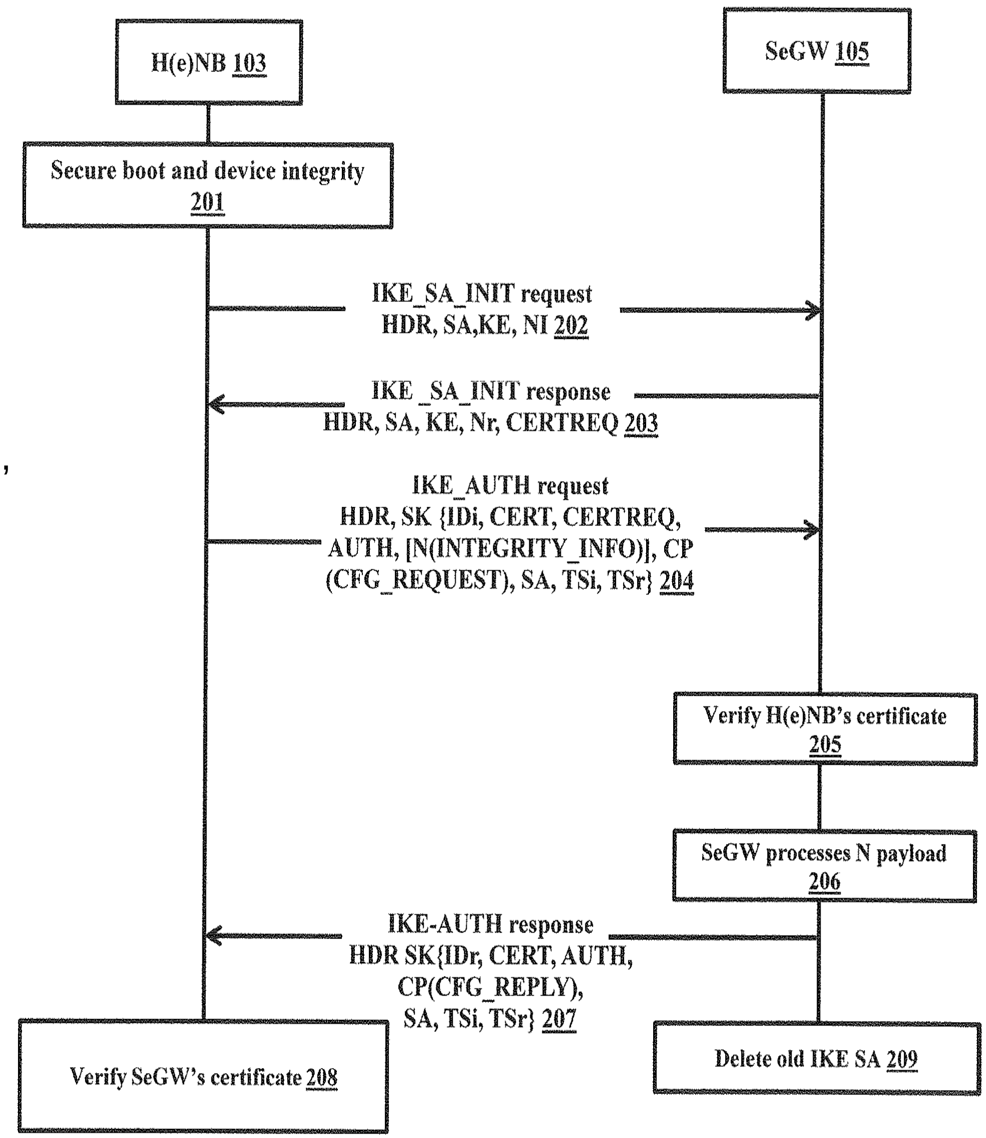
INTERNAL_ADDRESS(192.0.2.202)

INTERNAL_NETMASK(255.255.255.0)

INTERNAL_SUBNET(192.0.2.0/255.255.255.0)

TSi = (0, 0-65535,192.0.2.202-192.0.2.202)

TSr = (0, 0-65535,192.0.2.0-192.0.2.255)



IKEV2 --DELETE

INFORMATIONAL

Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump4

IKEV2 --OTHERS

- Error Handling
- IPComp
- ECN
- ....

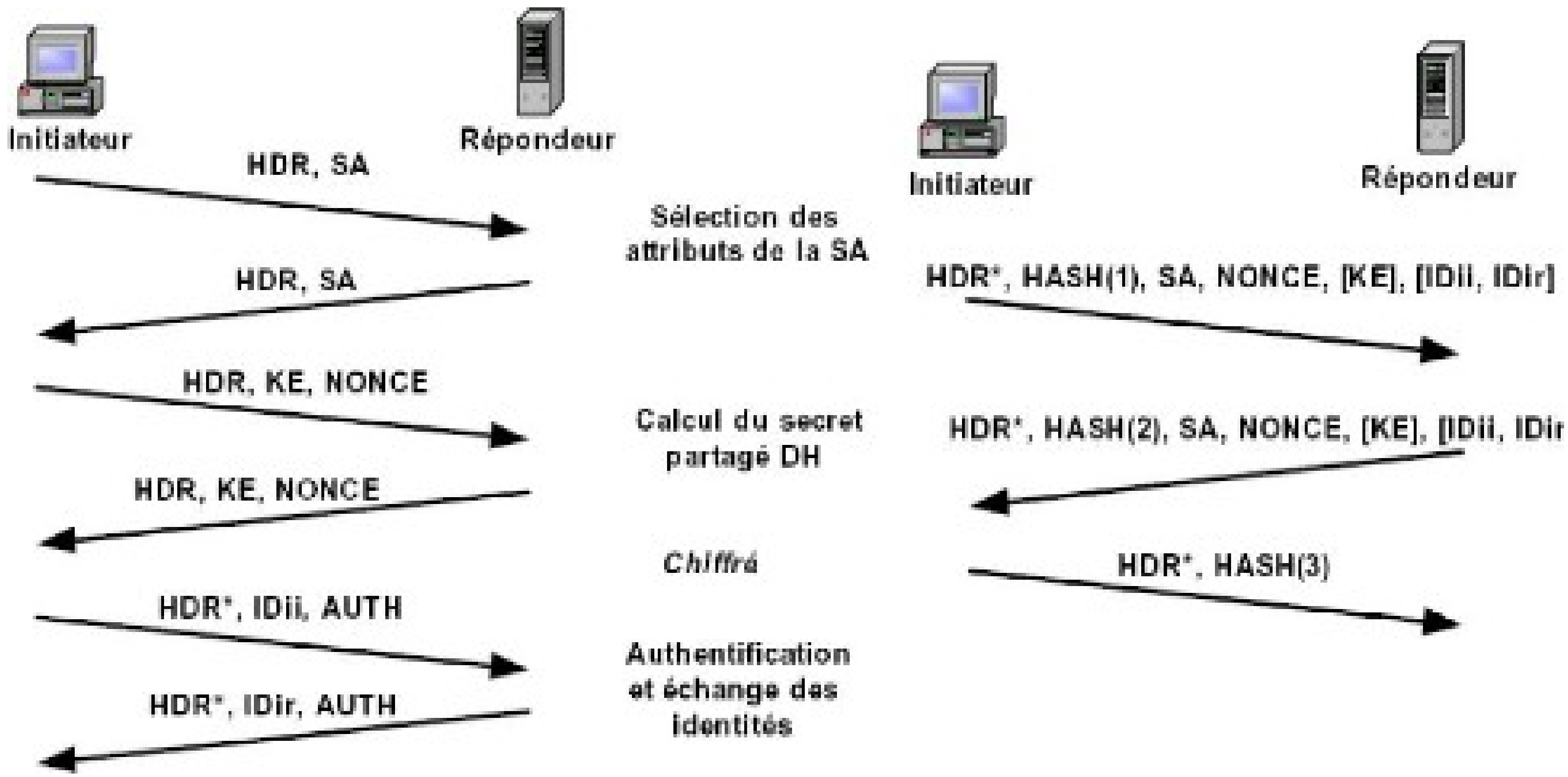
Packet format:
http://10.66.13.78/IKEv2/IKEv2_ENODE/4.html#koiPacketDump4

IKEV2 --LIBRESWAN

Configuration:

https://github.com/libreswan/libreswan/tree/master/testing/pluto/

IKEV1



Why is IKEv2 better than IKEv1?

http://netsecinfo.blogspot.com/2008/02/why-is-ikev2-better-than-ikev1.html

IKEV3

Why IKEv3 is better than v2/v1?

http://www.ietf.org/proceedings/85/slides/slides-85-ipsecme-7.pdf

RFC:

http://tools.ietf.org/html/draft-harkins-ikev3-00

Appendix A

```
echo "[LUCIEN]create CA key and cert"
openssl req -x509 -newkey rsa:1024 -out cacert.pem -keyout cakey.pem -days 3650
echo "[LUCIEN]create CA crl"
openssl ca -gencrl -out crl.pem -cert cacert.pem -keyfile cakey.pem
echo "==================================="
echo "[LUCIEN]create server KEY"
openssl genrsa -des3 -out serverKey.pem 1024
echo "[LUCIEN]create server req KEY"
openssl req -new -key serverKey.pem -out serverKey.csr
echo "[LUCIEN]create server CERT"
openssl ca -in serverKey.csr -out serverKey.crt -cert cacert.pem -keyfile cakey.pem
echo "==================================="
echo "[LUCIEN]create client KEY"
openssl genrsa -des3 -out clientKey.pem 1024
echo "[LUCIEN]create client req KEY"
openssl req -new -key clientKey.pem -out clientKey.csr
echo "[LUCIEN]create client CERT"
openssl ca -in clientKey.csr -out clientKey.crt -cert cacert.pem -keyfile cakey.pem
echo "==================================="
echo "[LUCIEN]import server cert"
openssl pkcs12 -export -in serverKey.crt -out serverKey.p12 -inkey serverKey.pem -name "serverKey.crt"
echo "[LUCIEN]import client cert"
openssl pkcs12 -export -in clientKey.crt -out clientKey.p12 -inkey clientKey.pem -name "clientKey.crt"
```

```
pk12util -i clientKey.p12 -d /etc/ipsec.d
pk12util -i serverKey.p12 -d /etc/ipsec.d
```

Appendix B

1. e , n 不能得出 d:

（1） ed≡1 (mod φ(n)) 。只有知道 e 和 φ(n) (Euclid) ，才能算出 d 。

（2） φ(n)=(p-1)(q-1) 。只有知道 p 和 q ，才能算出 φ(n) 。

（3） n=pq 。只有将 n 因数分解，才能算出 p 和 q 。

2. M^ed mod n =M:

$$a^{p-1} \equiv 1 \pmod{p}$$

$$ed \equiv 1 \pmod{(p-1)(q-1)} \qquad ed - 1 = h(p-1)(q-1)$$

$$m^{ed} = m^{(ed-1)}m = m^{h(p-1)(q-1)}m = \left(m^{p-1}\right)^{h(q-1)} m \equiv 1^{h(q-1)}m \equiv m \pmod{p}$$