CNT4406/5412 Network Security

IPsec/IKE

Zhi Wang

Florida State University

Fall 2014

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

- IKE v1 has some significant issues

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

- IKE v1 has some significant issues
  - ➠ specified in three separate RFCs: 2407, 2408, and 2409

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

- IKE v1 has some significant issues
  - specified in three separate RFCs: 2407, 2408, and 2409
  - complex and ambiguous in implementation details

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

- IKE v1 has some significant issues
  - specified in three separate RFCs: 2407, 2408, and 2409
  - complex and ambiguous in implementation details
  - NAT-unfriendly

Introduction

Internet Key Exchange is the protocol used to set up a security association (SA) in the IPsec protocol suite

- IKE v1 has some significant issues
  - specified in three separate RFCs: 2407, 2408, and 2409
  - complex and ambiguous in implementation details
  - NAT-unfriendly
- The authors hate IKE v1, one of them wrote a RFC for IKE v2

Introduction...

- IKE v2 is specified in RFC 5996, combining three RFCs for IKE v1

Introduction...

- IKE v2 is specified in RFC 5996, combining three RFCs for IKE v1
  - ➠ the leading author of RFC 5996 is Charlies Kaufman

Introduction...

- IKE v2 is specified in RFC 5996, combining three RFCs for IKE v1
  - the leading author of RFC 5996 is Charlies Kaufman
  - initially published in RFC 4306 with clarifications in RFC 4718

Introduction...

- IKE v2 is specified in RFC 5996, combining three RFCs for IKE v1
  - the leading author of RFC 5996 is Charlies Kaufman
  - initially published in RFC 4306 with clarifications in RFC 4718
  - more robust and cleaner than IKE v1

Introduction...

- IKE v2 is specified in RFC 5996, combining three RFCs for IKE v1
  - the leading author of RFC 5996 is Charlies Kaufman
  - initially published in RFC 4306 with clarifications in RFC 4718
  - more robust and cleaner than IKE v1
- We will cover IKE v2 exclusively
  - we use the term of "IKE" instead of "IKE v2" for clarity

Introduction to IKE v2

- IKE performs mutual authentication between two parties

Introduction to IKE v2

- IKE performs mutual authentication between two parties
- A IKE communication consists of a request and a response
  - ➠ the pair is called an IKE "exchange"

Introduction to IKE v2

- IKE performs mutual authentication between two parties
- A IKE communication consists of a request and a response
  - the pair is called an IKE "exchange"
  - every IKE message has a seq# to match requests and responses

Introduction to IKE v2

- IKE performs mutual authentication between two parties
- A IKE communication consists of a request and a response
  - the pair is called an IKE "exchange"
  - every IKE message has a seq# to match requests and responses
- The requester is responsible to ensure reliability (retransmission)

Introduction to IKE v2...

- IKE first establishes an IKE SA to secure IKE communication

Introduction to IKE v2...

- IKE first establishes an IKE SA to secure IKE communication
  - ➠ two exchanges: IKE_SA_INIT and IKE_AUTH

Introduction to IKE v2...

- IKE first establishes an IKE SA to secure IKE communication
  - ➠ two exchanges: IKE_SA_INIT and IKE_AUTH
- IKE then creates Child SAs for AH/ESP

Introduction to IKE v2...

- IKE first establishes an IKE SA to secure IKE communication
  - ➠ two exchanges: IKE_SA_INIT and IKE_AUTH
- IKE then creates Child SAs for AH/ESP
  - ➠ CREATE_CHILD_SA and INFORMATIONAL exchanges

Introduction to IKE v2...

- IKE first establishes an IKE SA to secure IKE communication
  - ➠ two exchanges: IKE_SA_INIT and IKE_AUTH
- IKE then creates Child SAs for AH/ESP
  - ➠ CREATE_CHILD_SA and INFORMATIONAL exchanges
- IKE and child SAs can be renegotiated during a session (rekeying)
  - ➠ to rekey a SA, create a replacement then delete the old one

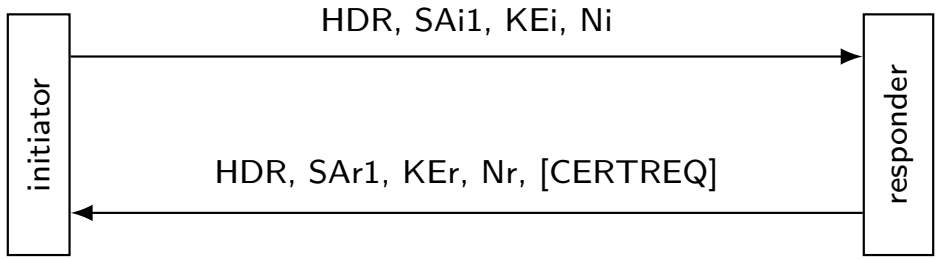
IKE Payload

IKE messages consist of a number of **payloads** linked by "next payload"

Notation	Payload
AUTH	authentication
CERT	certificate
CERTREQ	certificate request
D	delete
HDR	common IKE header (not a payload)
IDi/IDr	identification - initiator/responder
KE	DH key exchange
Ni, Nr	nonce -initiator/responder
N	notify
SA	security association
SK	encrypted and authenticated
TSi/TSr	traffic selector - initiator/ responder

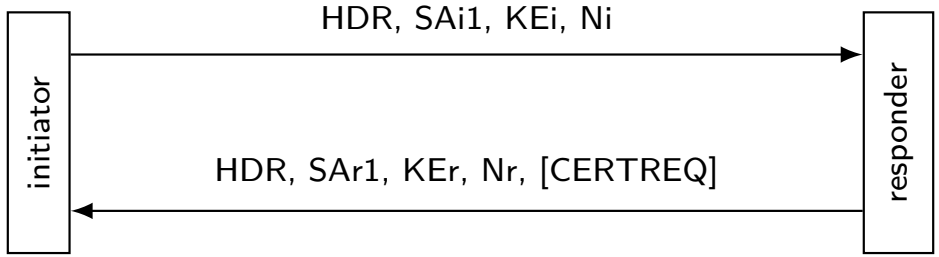
Initial Exchange: IKE_SA_INIT

- This exchange negotiates security parameters for IKE SA



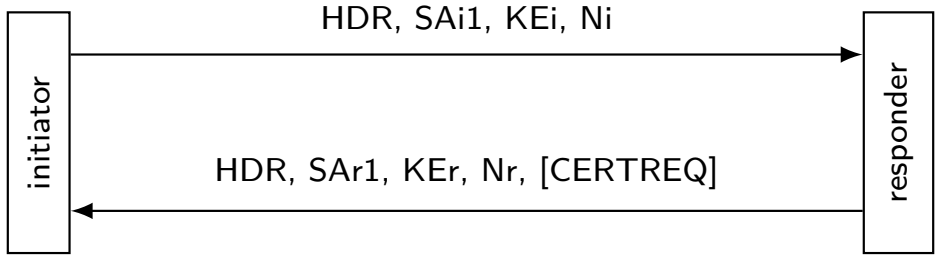
Initial Exchange: IKE_SA_INIT

- This exchange negotiates security parameters for IKE SA
  - SAi1 and SAr1 negotiate crytographic algorithms for IKE SA



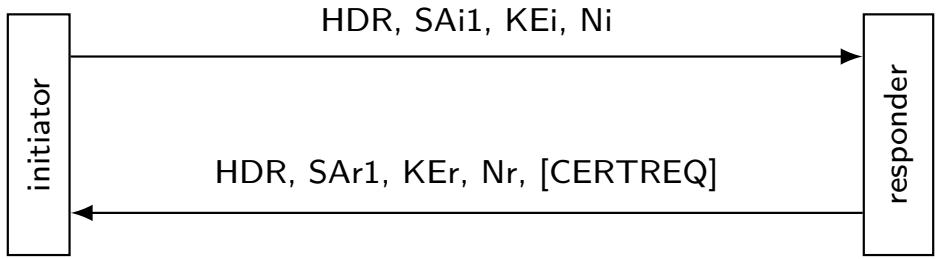
Initial Exchange: IKE_SA_INIT

- This exchange negotiates security parameters for IKE SA
  - SAi1 and SAr1 negotiate crytographic algorithms for IKE SA
  - KEi and KEr do a DH key exchange



Initial Exchange: IKE_SA_INIT

- This exchange negotiates security parameters for IKE SA
  - SAi1 and SAr1 negotiate crytographic algorithms for IKE SA
  - KEi and KEr do a DH key exchange
  - Ni and Nr are two nounces



Initial Exchange: IKE_SA_INIT...

- Each party can generate SKEYSEED from IKE_SA_INIT

*do not let initiator and responder do the same things

Initial Exchange: IKE_SA_INIT...

- Each party can generate SKEYSEED from IKE_SA_INIT
- All keys for IKE SA are derived from SKEYSEED

*do not let initiator and responder do the same things

Initial Exchange: IKE_SA_INIT...

- Each party can generate SKEYSEED from IKE_SA_INIT
- All keys for IKE SA are derived from SKEYSEED
  - ➠ each creates its own* SK_e (encryption) and SK_a (authentication)

*do not let initiator and responder do the same things

Initial Exchange: IKE_SA_INIT...

- Each party can generate SKEYSEED from IKE_SA_INIT
- All keys for IKE SA are derived from SKEYSEED
  - ➠ each creates its own* SK_e (encryption) and SK_a (authentication)
  - ➠ **SK** payloads are encrypted and authenticated with SK_e and SK_a

*do not let initiator and responder do the same things

Initial Exchange: IKE_SA_INIT...

- Each party can generate SKEYSEED from IKE_SA_INIT
- All keys for IKE SA are derived from SKEYSEED
  - ➠ each creates its own* SK_e (encryption) and SK_a (authentication)
  - ➠ **SK** payloads are encrypted and authenticated with SK_e and SK_a
- SKEYSEED also derives SK_d to be used for Child SAs

*do not let initiator and responder do the same things

Initial Exchange: IKE_SA_INIT...

Payload details:

Payload	Format
HDR	SPIi|SPIr|Next Payload|Ver|Exchange Type|Flag|Message ID|Length
KE	Next Payload|Payload Length|DH Group|DH Data
N	Next Payload|Length|Nounce Data

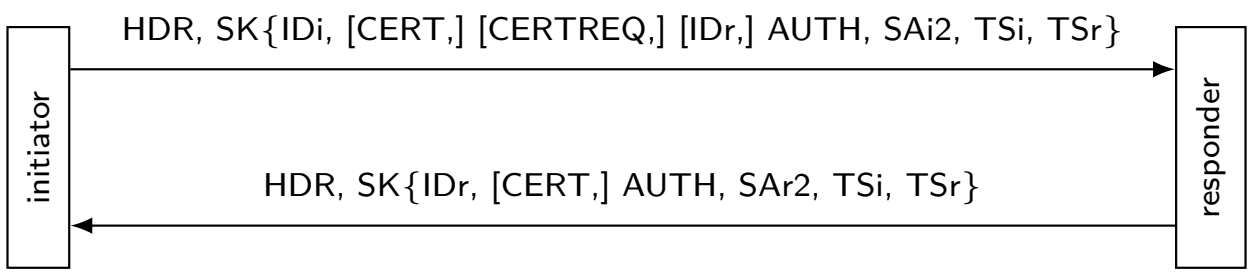
- Exchange type: `IKE_SA_INIT`, `IKE_AUTH`, `CREATE_CHILD_SA`, `INFORMATIONAL`
- SA negotiates the algorithms, it has a nested data structure

Initial Exchange: IKE_SA_INIT (ASCII Art)...

```
SA Payload
     |
     +--- Proposal #1 ( Proto ID = ESP(3), SPI size = 4,
     |     |            7 transforms,      SPI = 0x052357bb )
     |     |
     |     +-- Transform ENCR ( Name = ENCR_AES_CBC )
     |     |     +-- Attribute ( Key Length = 128 )
     |     |
     |     +-- Transform ENCR ( Name = ENCR_AES_CBC )
     |     |     +-- Attribute ( Key Length = 192 )
     |     |
     |     +-- Transform ENCR ( Name = ENCR_AES_CBC )
     |     |     +-- Attribute ( Key Length = 256 )
     |     |
     |     +-- Transform INTEG ( Name = AUTH_HMAC_SHA1_96 )
     |     +-- Transform INTEG ( Name = AUTH_AES_XCBC_96 )
     |     +-- Transform ESN ( Name = ESNs )
     |     +-- Transform ESN ( Name = No ESNs )
     |
     +--- Proposal #2 ( Proto ID = ESP(3), SPI size = 4,
           |            4 transforms,      SPI = 0x35a1d6f2 )
           |
           +-- Transform ENCR ( Name = AES-GCM with a 8 octet ICV )
           |     +-- Attribute ( Key Length = 128 )
           |
           +-- Transform ENCR ( Name = AES-GCM with a 8 octet ICV )
           |     +-- Attribute ( Key Length = 256 )
           |
           +-- Transform ESN ( Name = ESNs )
           +-- Transform ESN ( Name = No ESNs )
```

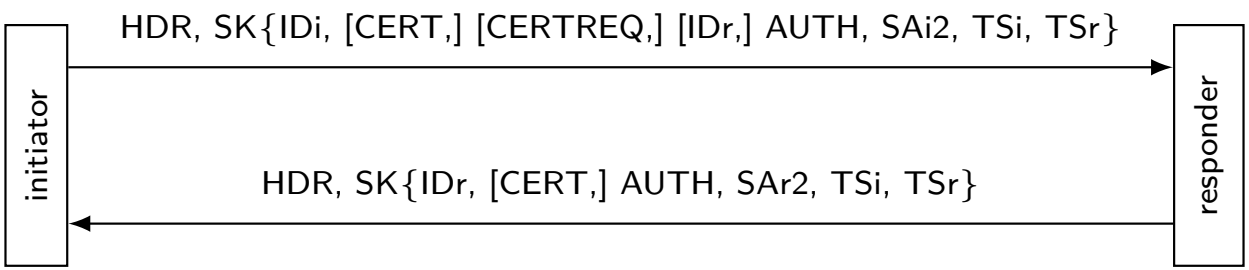
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected



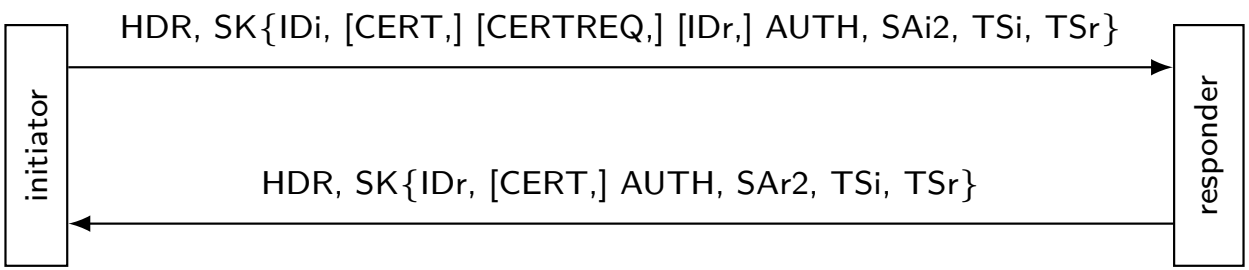
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)



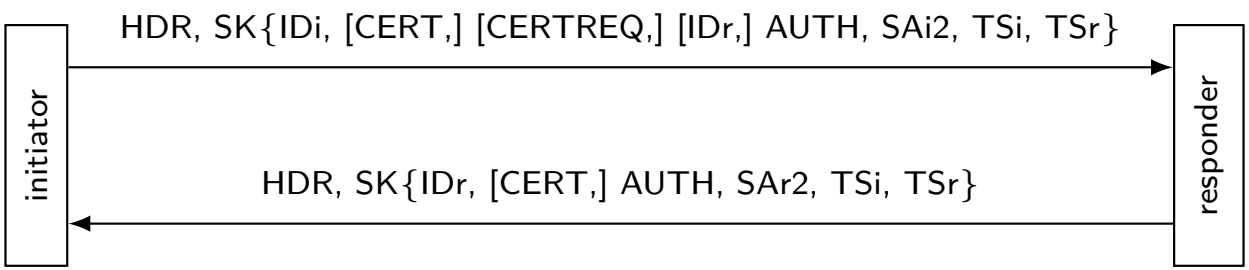
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...''), < InitiatorSignedBytes >)$



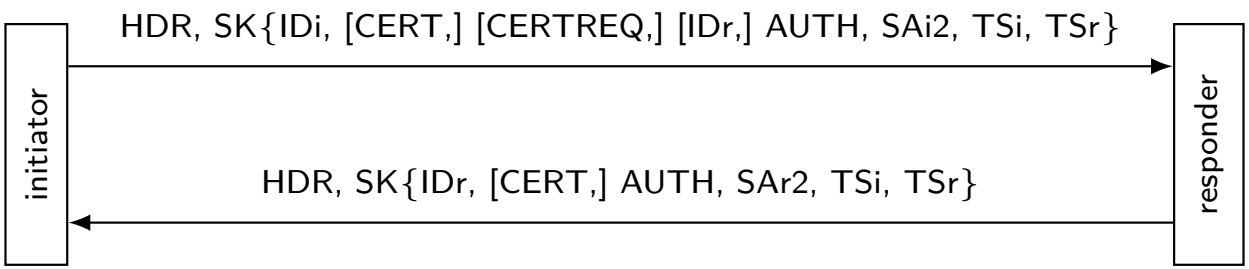
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...''), < InitiatorSignedBytes >)$
  - to prevent man-in-the-middle attack



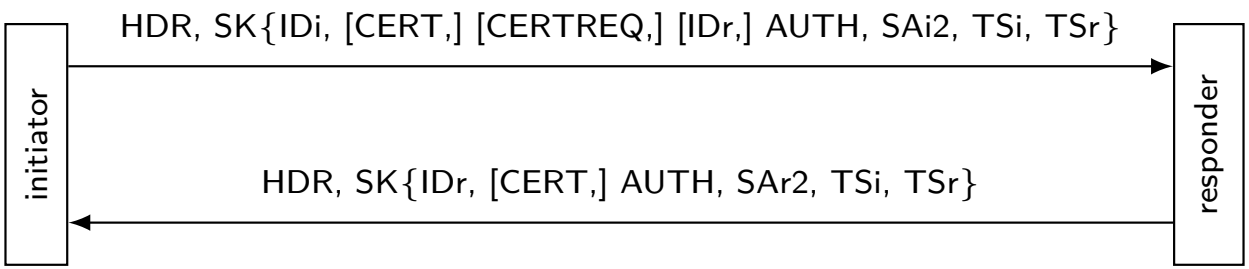
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...\text{”}), < InitiatorSignedBytes >)$
  - to prevent man-in-the-middle attack
- It exchanges identities and certificates (IDi/IDr)



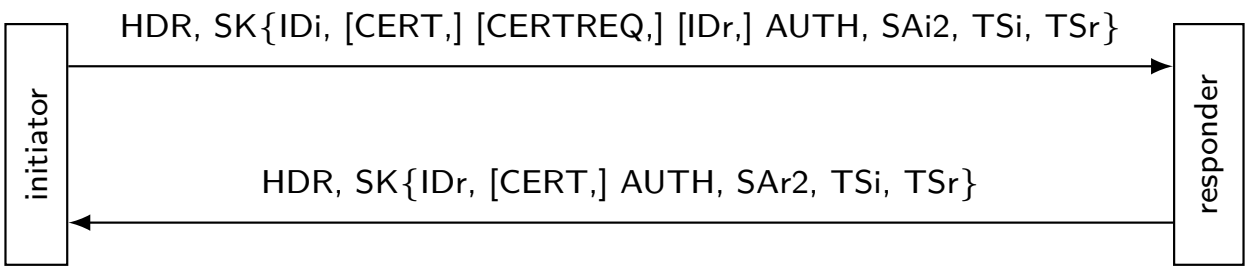
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...\text{”}), < InitiatorSignedBytes >)$
  - to prevent man-in-the-middle attack
- It exchanges identities and certificates (IDi/IDr)
- It establishes the first Child SA (SAi2/SAr2)



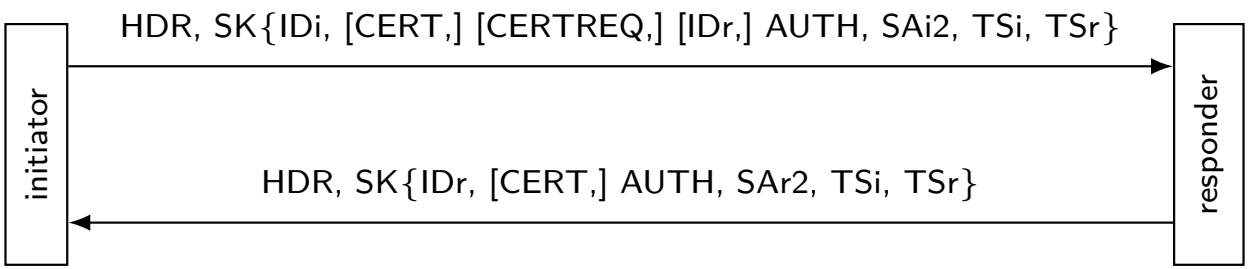
Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - ➠ e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...\text{”}), < InitiatorSignedBytes >)$
  - ➠ to prevent man-in-the-middle attack
- It exchanges identities and certificates (IDi/IDr)
- It establishes the first Child SA (SAi2/SAr2)
  - ➠ SAi2/SAr2 and TSi/TSr are used as key materials for Child SAs



Initial Exchange: IKE_AUTH

- IKE_AUTH messages are encrypted and integrity protected
- It authenticates previous messages of IKE_SA_INIT (AUTH)
  - e.g., $AUTH = prf(prf(shared\ \ secret, \text{“}Key\ \ Pad...\text{”}), < InitiatorSignedBytes >)$
  - to prevent man-in-the-middle attack
- It exchanges identities and certificates (IDi/IDr)
- It establishes the first Child SA (SAi2/SAr2)
  - SAi2/SAr2 and TSi/TSr are used as key materials for Child SAs
  - failure to negotiate Child SA does not invalid IKE SA



Initial Exchange: IKE_AUTH...

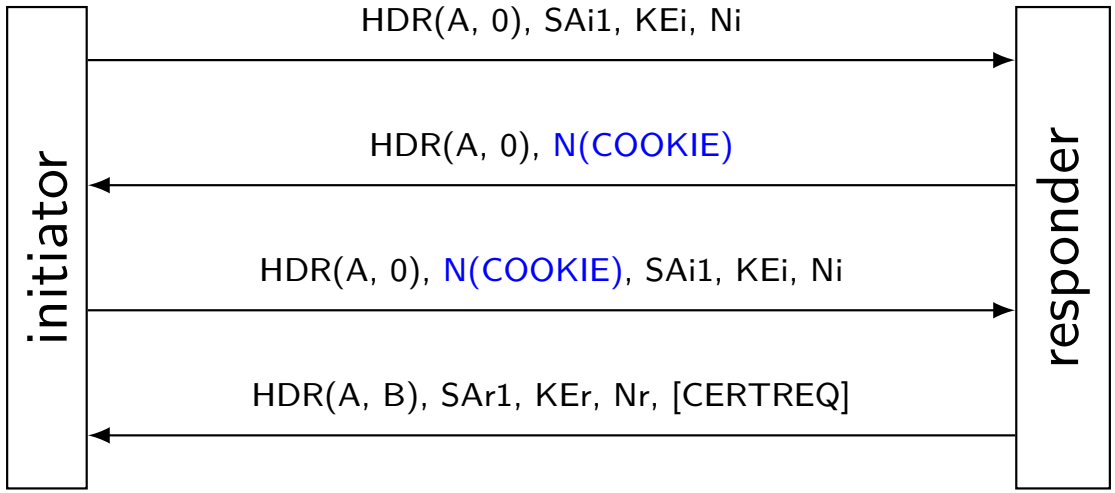
Payload details:

Payload	Format
SK	Next Payload|Payload Length|IV|Encrypted IKE Payloads|MAC
ID	Next Payload|Payload Length|ID Type|ID Data
AUTH	Next Payload|Payload Length|Auth Method|Auth Data
TS	Next Payload|Payload Length|Number of TSs|Selectors

- SK is an encrypted container of other IKE payloads, it must be the last payload in a IKE message
- ID can be IPv4 address, domain name, IPv6 address...
- Traffic selector has src/dst IP rangers, IP protocol, and a port range

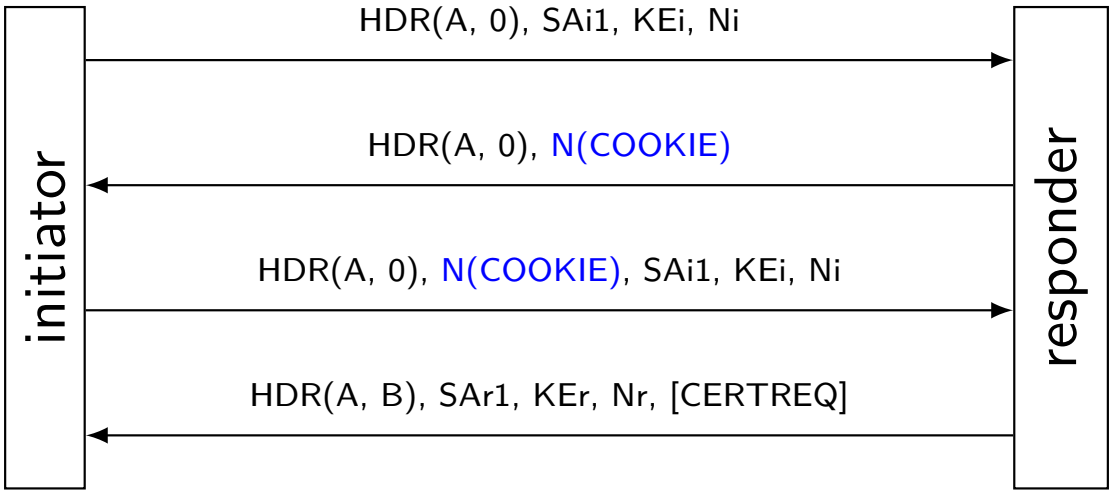
Initial Exchange: Denial-of-Service Protection

- It is only enabled when a responder detects many half-open IKE SAs



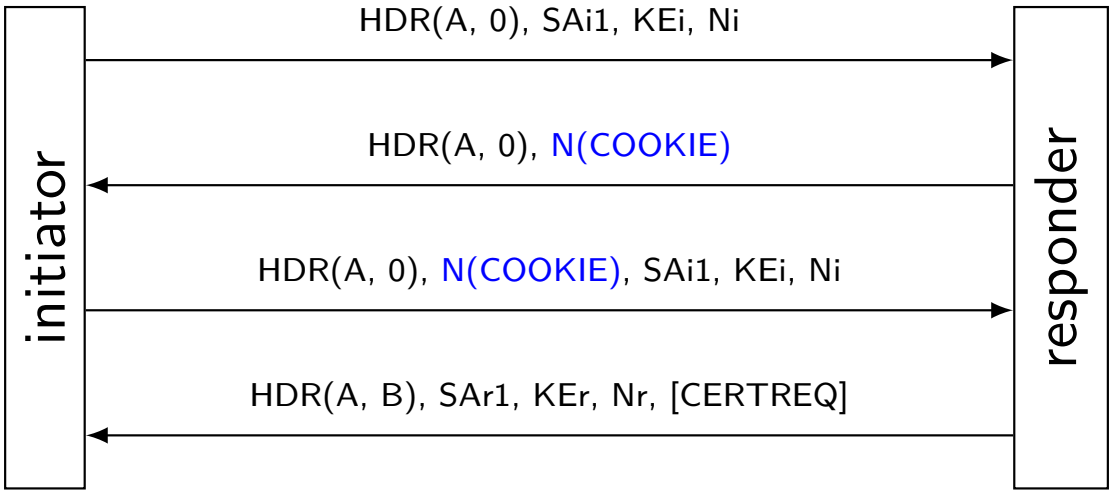
Initial Exchange: Denial-of-Service Protection

- It is only enabled when a responder detects many half-open IKE SAs
- IKE uses stateless cookie for DoS protection



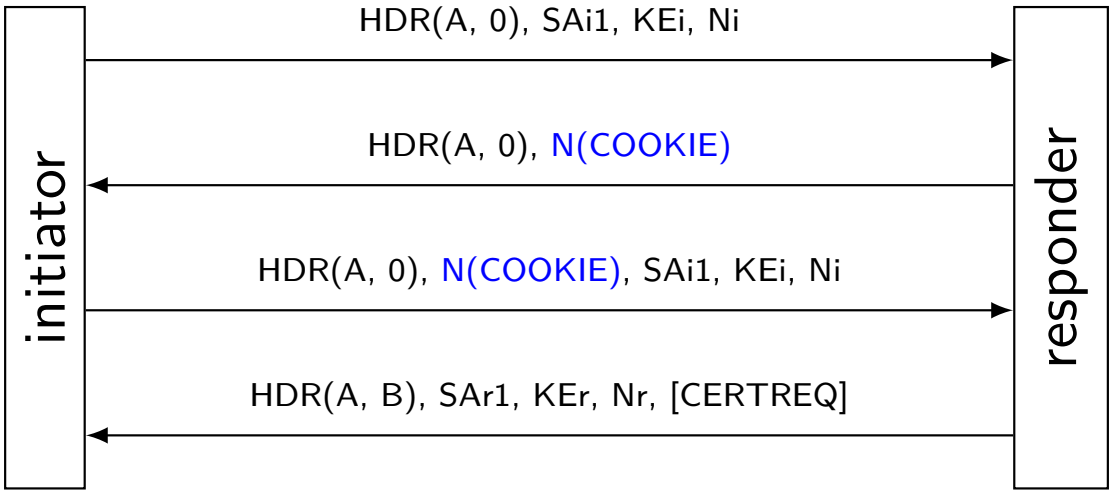
Initial Exchange: Denial-of-Service Protection

- It is only enabled when a responder detects many half-open IKE SAs
- IKE uses stateless cookie for DoS protection
  - RFC5996: $Cookie = version\ of\ secret|Hash(Ni|IPi|SPIi|secret)$



Initial Exchange: Denial-of-Service Protection

- It is only enabled when a responder detects many half-open IKE SAs
- IKE uses stateless cookie for DoS protection
  - RFC5996: $Cookie = version\ of\ secret | Hash(Ni|IPi|SPIi|secret)$
  - the secret for DoS protection is changed periodically, why?



Initial Exchange

Does IKE initial exchange has...

- perfect forward secrecy?

Initial Exchange

Does IKE initial exchange has...

- perfect forward secrecy?
- denial-of-service protection?

Initial Exchange

Does IKE initial exchange has...

- perfect forward secrecy?
- denial-of-service protection?
- endpoint identifier hiding?

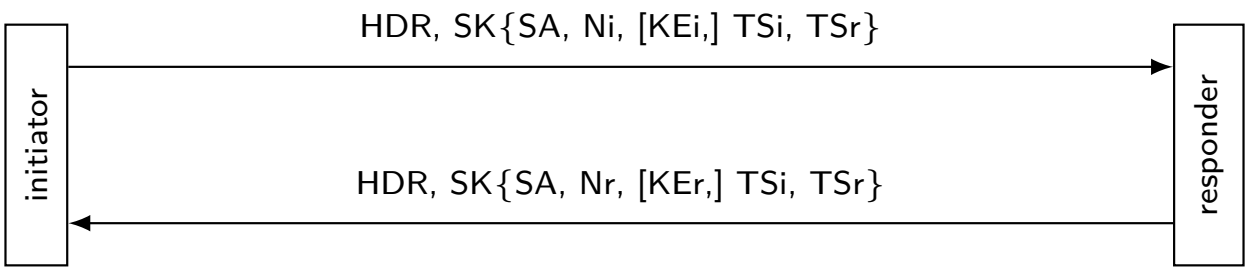
Initial Exchange

Does IKE initial exchange has...

- perfect forward secrecy?
- denial-of-service protection?
- endpoint identifier hiding?
- live partner reassurance?

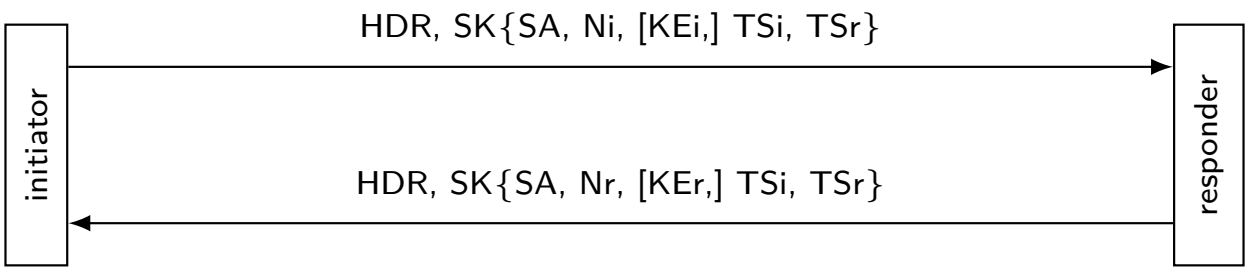
CREATE_CHILD_SA Exchange

- CREATE_CHILD_SA exchange is used to create new Child SAs



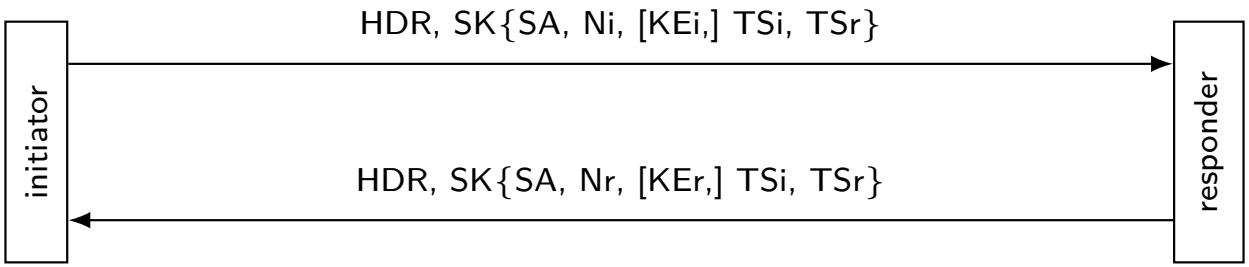
CREATE_CHILD_SA Exchange

- CREATE_CHILD_SA exchange is used to create new Child SAs
- It can also be used to rekey both IKE_SAs and Child_SAs



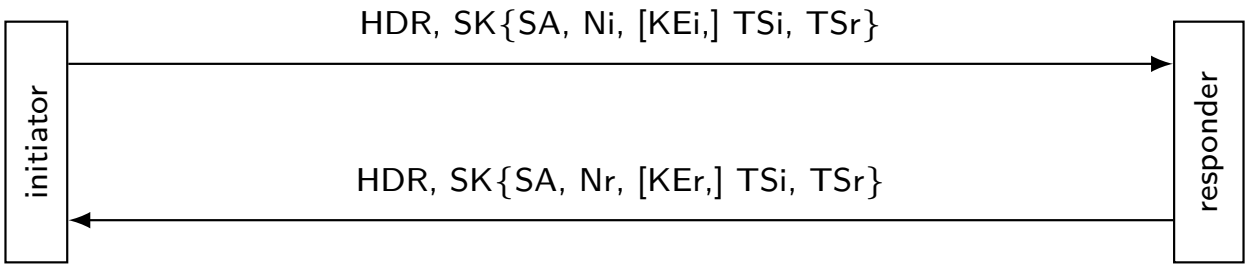
CREATE_CHILD_SA Exchange

- CREATE_CHILD_SA exchange is used to create new Child SAs
- It can also be used to rekey both IKE_SAs and Child_SAs
  - ➠ an SA is rekeyed by creating a new SA and deleting the old one



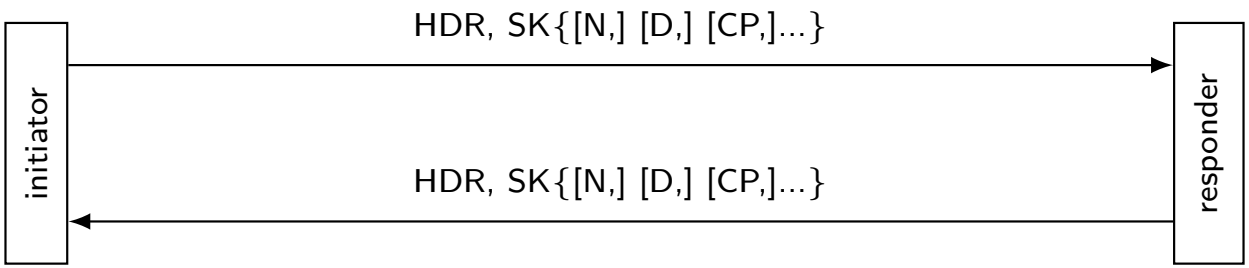
CREATE_CHILD_SA Exchange

- CREATE_CHILD_SA exchange is used to create new Child SAs
- It can also be used to rekey both IKE_SAs and Child_SAs
  ➠ an SA is rekeyed by creating a new SA and deleting the old one
- It can be initiated by either party of the IKE SA after initial exchange



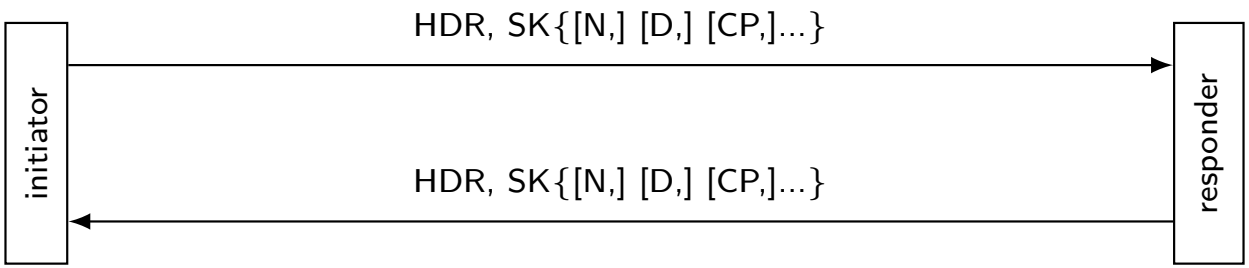
INFORMATIONAL Exchange

- INFORMATIONAL exchange is used to convey control messages



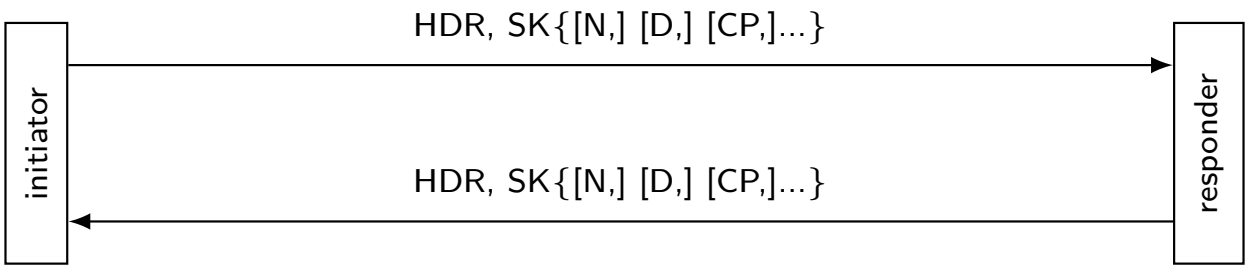
INFORMATIONAL Exchange

- INFORMATIONAL exchange is used to convey control messages
  - notification, delete, and configuration payloads



INFORMATIONAL Exchange

- INFORMATIONAL exchange is used to convey control messages
  - ➠ notification, delete, and configuration payloads
- It can be initiated by either party of the IKE SA after initial exchange



NAT Traversal

- NAT needs to modify the IP/TCP/UDP header
  - ➠ e.g., src/dst IP address, TCP/UDP ports

NAT Traversal

- NAT needs to modify the IP/TCP/UDP header
  ➠ e.g., src/dst IP address, TCP/UDP ports
- Fields changed by NAT may be protected by IPsec
  ➠ e.g., src IP address is protected by AH, and included in the TCP/UDP checksum (encrypted in ESP)

NAT Traversal

- NAT needs to modify the IP/TCP/UDP header
  ➠ e.g., src/dst IP address, TCP/UDP ports
- Fields changed by NAT may be protected by IPsec
  ➠ e.g., src IP address is protected by AH, and included in the TCP/UDP checksum (encrypted in ESP)
- Solution: encapsulating IKE/ESP in a UDP packet (port 4500)

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header
- IKE will switch to UDP port 4500 when NAT is detected (how?)

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header
- IKE will switch to UDP port 4500 when NAT is detected (how?)
  - ➠ to detect NAT, ask peer to send back observed my address/port

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header
- IKE will switch to UDP port 4500 when NAT is detected (how?)
  ➠ to detect NAT, ask peer to send back observed my address/port
- ESP will also be encapsulated in UDP port 4500 when NAT exists

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header
- IKE will switch to UDP port 4500 when NAT is detected (how?)
  - ➠ to detect NAT, ask peer to send back observed my address/port
- ESP will also be encapsulated in UDP port 4500 when NAT exists
  - ➠ IP addresses for TCP/UDP checksum come from traffic selector

NAT Traversal...

- IKE normally uses UDP port 500, IKE header follows the UDP header
- IKE will switch to UDP port 4500 when NAT is detected (how?)
  - ➠ to detect NAT, ask peer to send back observed my address/port
- ESP will also be encapsulated in UDP port 4500 when NAT exists
  - ➠ IP addresses for TCP/UDP checksum come from traffic selector
  - ➠ IKE messages are prepended by four octets of zero to distinguish it from ESP messages (how?)

Summary

- IKE History
- IKE Payloads
- IKE Exchanges
- DoS Protection and NAT Traversal

- Next lecture: SSL/TLS