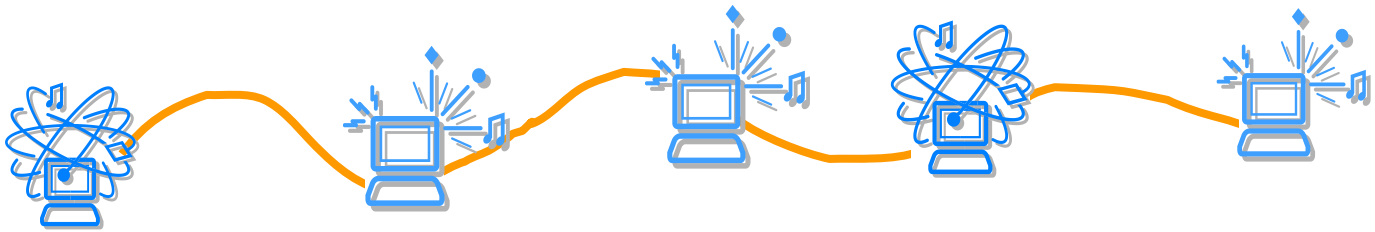
Segurança em Redes
VPN – IKEv2



Redes de Comunicação
Área Departamental de Engenharia da Electrónica e Telecomunicações e de Computadores
Instituto Superior de Engenharia de Lisboa

Internet Key Exchange
IKEv2 'bis' – RFC 7296

Introduction

IP Security (IPsec) provides confidentiality, data integrity, access control, and data source authentication to IP datagrams.

These services are provided by maintaining shared state between the source and the sink of an IP datagram. This state defines, among other things, the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms.

IKE performs mutual authentication between two parties and establishes an IKE Security Association (SA) that includes shared secret information that can be used to efficiently establish SAs for Encapsulating Security Payload (ESP) [ESP] or Authentication Header (AH) [AH] and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry.

Introduction (cont.)

The protocol described in this document retains the same major version number (2) and minor version number (0) as was used in RFC 4306. That is, the version number is *not* changed from RFC 4306.

The small number of technical changes listed here are not expected to affect RFC 4306 implementations that have already been deployed at the time of publication of this document.

Sumário

- Motivações para a gestão de chaves
- Conceitos principais
    - Protocolo Diffie-Hellman
    - *Perfect Forward Secrecy*
    - Função pseudo-aleatória (PRF)

- IKEv2
    - Geração de chaves de cifra e autenticação
    - Negociação de algoritmos criptográficos
    - Regeneração de chaves

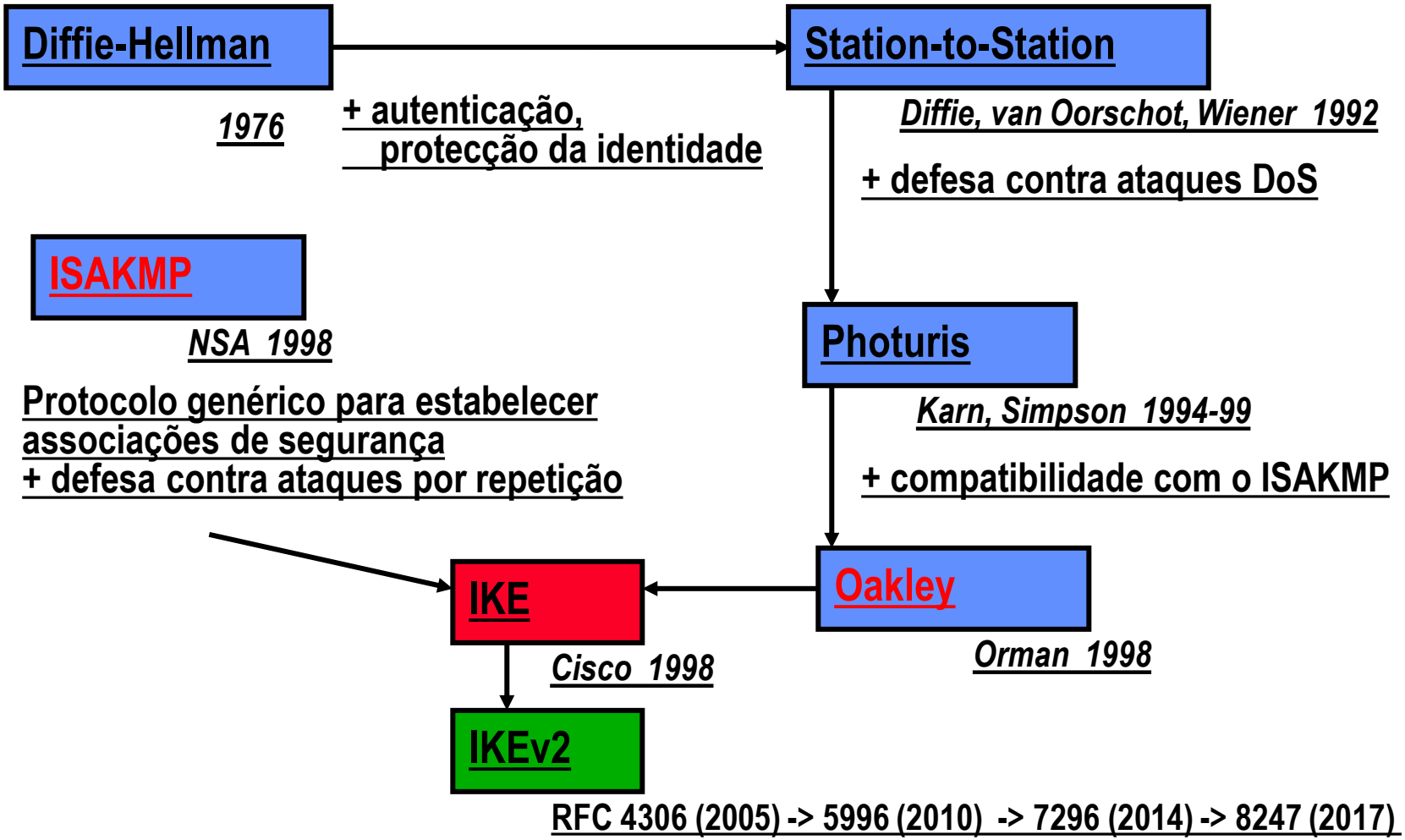
RFCs

- RFCs 2401 - 2412 (a **maioria obsoleta** pelos RFCs 43XX)
- **RFC 2409 (The Internet Key Exchange (IKE))** (Novembro 1998)
- RFC 3740 (*The Multicast Group Security Architecture*)
- RFC 4301 (*Security Architecture for the Internet Protocol*)
- RFC 4302 (IP Authentication Header)
- RFC 4303 (IP Encapsulating Security Payload (ESP))
- RFC 4304 (Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP))
- RFC 4305 (Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH))
- **RFC 4306 (Internet Key Exchange (IKEv2) Protocol)** (Dezembro 2005)
- RFC 4307 (Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2))
- RFC 4308 (Cryptographic Suites for IPsec)

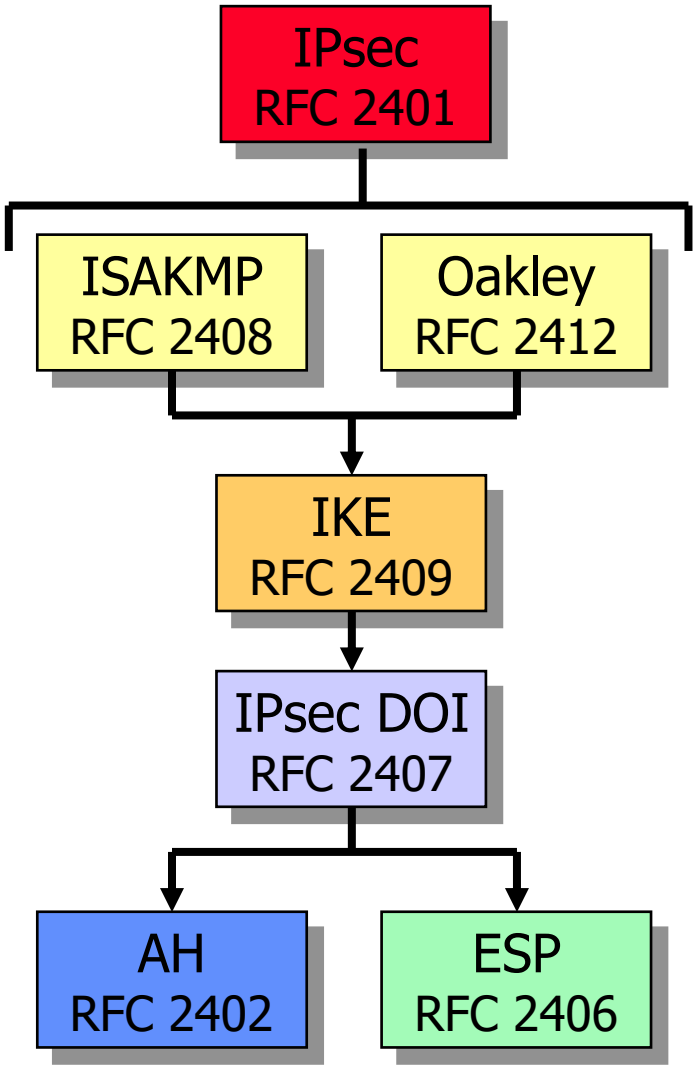
RFCs (cont.)

- RFC 4309 (Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP))
- RFC 4312 (The Camellia Cipher Algorithm and Its Use With IPsec)
- RFC 4718 (IKEv2 Clarifications and Implementation Guidelines)
- RFC 5288 (Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol)
- **RFC 5996 (Internet Key Exchange Protocol Version 2 (IKEv2))** – substituiu os dois RFC anteriores (Setembro 2010)
- RFC 5998 (Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol)
- RFC 6998 (Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2))
- **RFC 7296 (Internet Key Exchange Protocol Version 2 (IKEv2))** (IKEv2bis) (Outubro 2014)
- RFC 7427 (Signature Authentication in the Internet Key Exchange Version 2 (IKEv2))
- RFC 8247 (Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2))
- RFC 7815 (Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation) (Março 2016)

Genealogia do IKE



RFCs mais importantes (início/obsoletos)

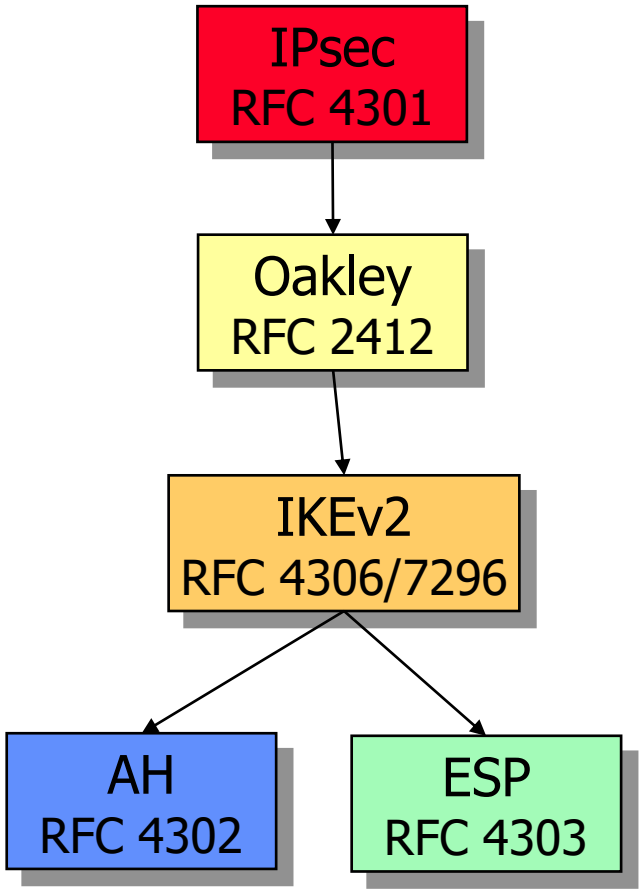


O IKEv2 não é compativel com o IKEv1

ISAKMP	Internet Security Association and Key Management Protocol
IKE	Internet Key Exchange
DOI	Domain of Interpretation
AH	Authentication Header
ESP	Encapsulating Security Payload

RFCs mais importantes

O IKEv2 não é compatível com o IKEv1



ISAKMP	Internet Security Association and Key Management Protocol
IKE	Internet Key Exchange
DOI	Domain of Interpretation
AH	Authentication Header
ESP	Encapsulating Security Payload

IKE: Resumo

- **Objectivo**: Criar uma associação de segurança (SA) entre dois *hosts*
  - Chaves partilhadas de autenticação e cifra e acordo sobre os algoritmos de cifra

- **Duas fases**:
  - 1ª fase estabelece uma associação de segurança (IKE SA) recorrendo a:
    - Diffie-Hellman (DH)
    - *Nonces* novos e a valores antigos obtidos pelo DH (adeus PFS!)
  - 2ª fase cria as *child* SA necessárias ao suporte de uma ou mais ligações entre as entidades.

IKE: Objectivos de segurança

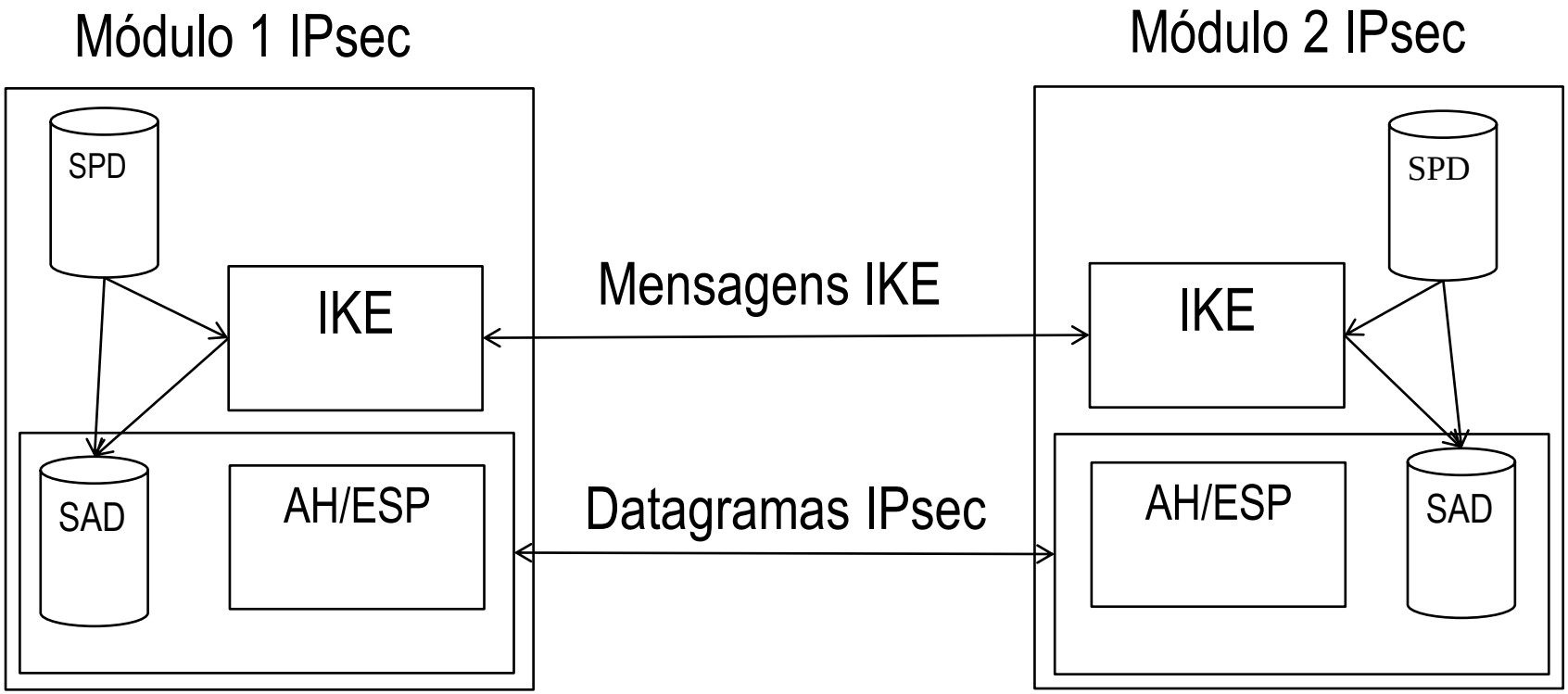
- Autenticação de entidades das partes participantes
- Estabelecimento de uma chave secreta partilhada principal.
- Chave secreta principal é usada para derivar outras chaves
  - Para confidencialidade e autenticação do canal de gestão do IKE
  - Para uso geral dos SA.
- Resistência a ataques *Denial-of-Service* (DoS)
  - Usando mecanismos de *cookies*.
- Negociação segura de todos os algoritmos
  - Protocolo de autenticação e/ou cifra, método de troca de chaves, grupo Diffie-Hellman, algoritmos para encriptação e de autenticação/integridade.
- Opções para ***Perfect Forward Secrecy***, ***Deniable Authentication*** e ***Identity Protection*.**

Porquê um desenho com duas fases?

- A 1ª fase, mais dispendiosa devido à necessidade de criação e troca de valores para o DH, cria o SA principal (SA_IKE).

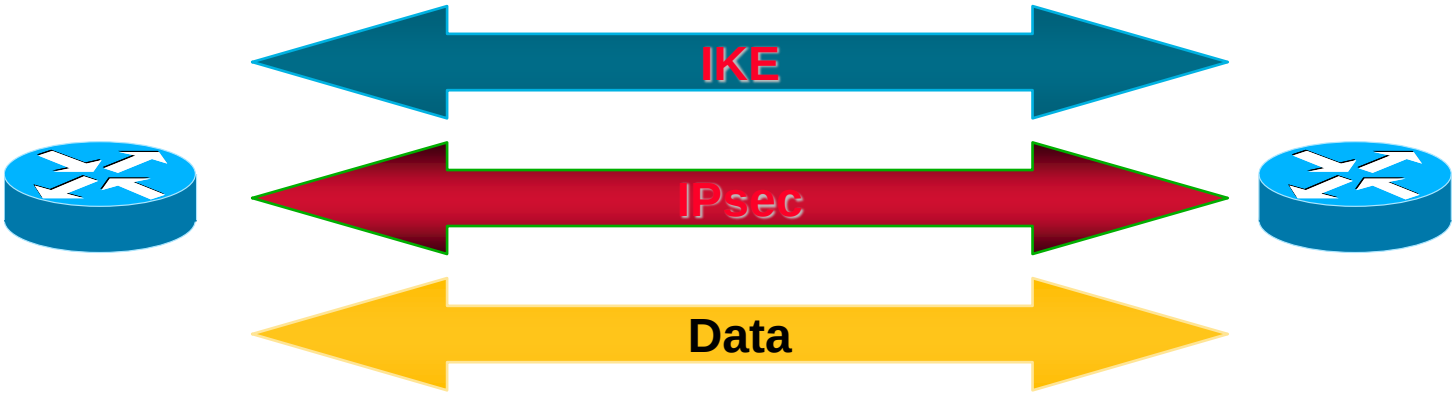
- A 2ª fase, menos dispendiosa, cria múltiplos *child* SA principais entre pares de extremos da ligação.
  - Conversações distintas podem necessitar de protecções diferentes
    - Algum tráfego apenas necessita protecção de integridade ou cifras fracas
    - É muito dispendioso utilizar sempre a segurança mais forte
  - Permite evitar que se multiplexem conversações sobre o mesmo *child* SA
  - *Child* SA distintos para diferentes classes de serviço.

Arquitectura IPsec



IKE: *Internet Key Exchange*
SAD: *Security Association Database*
SPD: *Security Policy Database*

Sequência para iniciar novas ligações IPsec

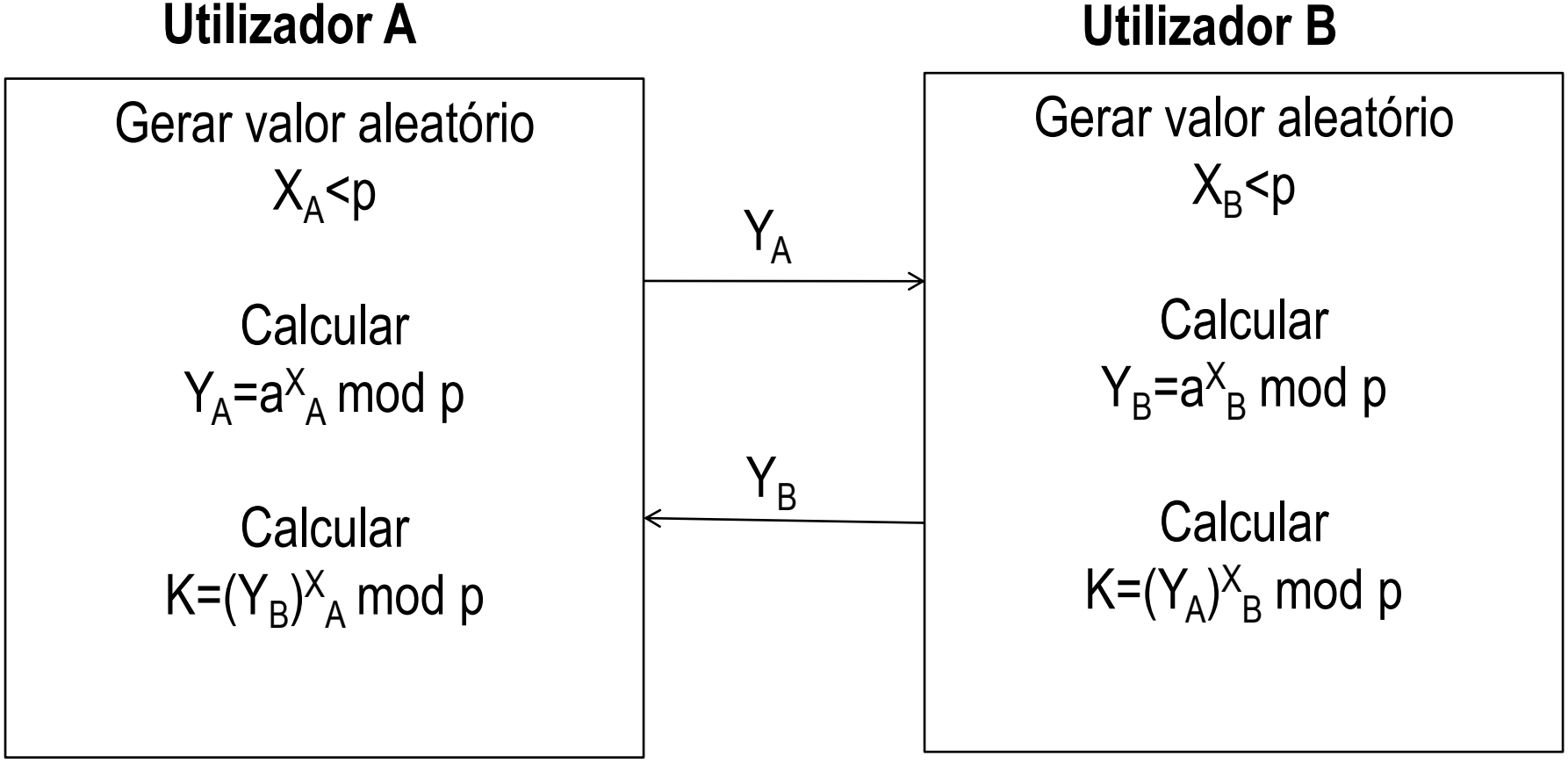


- Estabelecer os SA IKE
- Estabelecer os SA IPsec
  - Possíveis múltiplos SA IPsec por cada IKE SA
- Envio de dados protegidos

Porquê a gestão automática de chaves

- Necessário configurar chaves para o AH e o ESP
- Técnicas manuais
  - Mais simples
  - Práticas em ambientes pequenos e estáticos
  - A intervenção humana leva facilmente a enganos
  - Não escala bem
  - As chaves estáticas não se adequam bem à segurança

Protocolo de troca de chaves Diffie-Hellman



a e *p* não necessitam ser secretos podendo ser pré-calculados. São definidos pelos grupos Diffie-Hellman (RFCs 2409, 2631) de acordo com a dimensão pretendida para o primo *p*.

Diffie-Hellman na prática (Grupos Diffie-Hellman)

- Grupos "*Modular Exponential* (MODP) *Diffie-Hellman*" para troca de chaves no IKE:
    - 768-bit modulus and primitive root 2.
    - 1024-bit modulus and primitive root 2.
    - Parâmetros "Two elliptic curve DH"
    - 1536-bit MODP
    - 2048-bit MODP
    - 3072-bit MODP
    - 4096-bit MODP
    - 6144-bit MODP
    - 8192-bit MODP

Perfect Forward Secrecy (PFS)

- Diz respeito à noção de que o comprometimento de uma chave de sessão não compromete as outras chaves de sessão.

- Nenhuma chave pode ser usada repetidamente para fins distintos.

Para existir PFS nenhuma chave deve ser gerada a partir de uma chave anterior.

Função pseudo-aleatória (PRF)

- A função pseudo-aleatória recebe um chave de dimensão variável, dados de dimensão variável e produz uma saída de dimensão fixa .

- Utilizada para gerar chaves para o IKE a partir de HMAC.

- O RFC4307 recomenda:
  - PRF_HMAC_SHA1 **MUST** RFC2104
  - PRF_HMAC_MD5 **MAY** RFC2104
  - PRF_AES128_CBC **SHOULD+** AES-PRF

Uso das funções pseudo-aleatórias

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

onde:

T1 = prf (K, S | 0x01)

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

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

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

- As chaves necessárias (*SK_x*) são obtidas dos valores T1, T2, etc. gerados iterativamente pela função prf e concatenados:

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

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

Protocolo IKEv2

- **Define a forma de autenticação mútua entre duas partes e estabelece uma associação de segurança IKE (SA_IKE).**

  - **O SA_IKE** inclui informação sobre o segredo partilhado que pode ser usado para estabelecer de forma eficiente os SA para:
    - *Encapsulating Security Payload* (ESP) [RFC4303] e/ou
    - *Authentication Header* (AH) [RFC4302]

- Inclui o conjunto de algoritmos criptográficos a serem usados para proteger o tráfego.

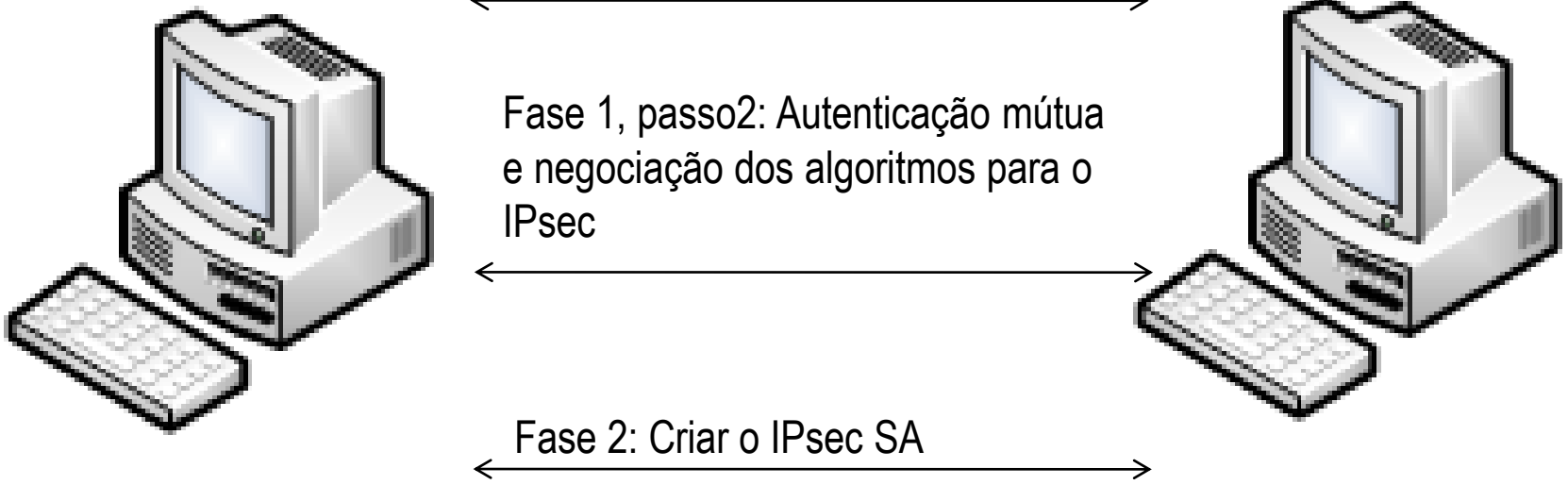
Protocolo IKEv2

- **Fase1, passo 1: IKE_SA_INIT**
  - Negociar os algoritmos IKE
  - Calcular as chaves secretas para o IKE
  - Calcular a chave SK_d que servirá para calcular as chaves para o IPsec na fase 2.

- **Fase 1, passo 2: IKE_AUTH**
  - Autenticação mútua
  - Negociação dos algoritmos IPsec (*piggybacked*).

- **Fase 2: CREAT_CHILD_SA**
  - Estabelecer as associações (SA) do AH e do ESP.

Resumo do IKEv2



Algoritmos criptográficos

- No contexto do IKE_SA são negociados quatro algoritmos criptográficos:
  - algoritmo de encriptação,
  - algoritmo de autenticação/integridade,
  - grupo Diffie-Hellman, e
  - função geradora de valores pseudo-aleatórios (prf).

- A função geradora pseudo-aleatória é utilizada na criação chaves para todos os algoritmos criptográficos utilizados, quer pelo IKE_SA, quer pelos CHILD_SA.

- Existem **IKE SA** e ***child* SA**.

Suites

- O termo ***suite*** refere-se a um conjunto completo de algoritmos usados para protege uma ligação e definidos nas SA.

- Um *initiator* propõe uma ou mais *suites* listando os algoritmos suportados que podem ser combinados.

- O IKE também pode negociar compressão (IPcomp), para além do AH e do ESP.

Mensagens IKE

- Todas as comunicações IKE consistem na **troca de pares de mensagens**: Pedido/Resposta.

- As primeiras mensagens para o estabelecimento do IKE SA são uma troca **IKE_SA_INIT,** seguida de uma troca **IKE_AUTH**, as mensagens seguintes são trocas: **CREATE_CHILD_SA** ou **INFORMATIONAL**.

- Normalmente existe apenas uma troca de mensagens IKE_SA_INIT e outra IKE_AUTH (total de 4 mensagens, 2 pares) para criar o IKE SA e o primeiro *child* SA.
  - Em casos excepcionais pode haver mais de uma troca deste tipo de mensagens.
  - A troca IKE_SA_INIT tem de terminar antes de passar à troca IKE_AUTH, a qual tem de terminar antes de poder haver mensagens CREATE_CHILD_SA ou INFORMATIONAL

Mensagens IKE

- A primeira troca de mensagens IKE_SA_INIT negoceia os parâmetros de segurança para o IKE SA, envia *nonces* e valores Diffie-Hellman.

- A segunda troca de mensagens IKE_AUTH autentica as mensagens trocadas anteriormente, troca identidades e certificados e cria um *child* SA para a primeira ligação a estabelecer com AH e/ou ESP. Parte desta mensagem é cifrada e a integridade protegida usando as chaves "trocadas" no IKE_SA_INIT, desta forma as identidades são escondidas dos bisbilhoteiros e todos os campos em todas as mensagens são autenticados.

- A mensagem INFORMATIONAL vazia pode servir de "keep alive".

Descriptions of the payloads contained in the messages

- AUTH Authentication
- CERT Certificate
- CERTREQ Certificate Request
- CP Configuration
- D Delete
- EAP Extensible Authentication
- HDR IKE Header (not a payload)
- IDi Identification - Initiator
- IDr Identification - Responder
- KE Key Exchange
- Ni, Nr Nonce
- N Notify
- SA Security Association
- SK Encrypted and Authenticated
- TSi Traffic Selector - Initiator
- TSr Traffic Selector - Responder
- V Vendor ID

Note: Right *initiator* (i), left *responder* (r).

IKE messages

- The **first exchange** of an IKE session, **IKE_SA_INIT**, negotiates security parameters for the IKE SA, sends nonces, and sends Diffie-Hellman values.

- The **second exchange**, **IKE_AUTH**, authenticate the previous messages, exchange identities and certificates, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH or ESP Child SA (unless there is failure setting up the AH or ESP Child SA, in which case the IKE SA is still established without the Child SA).

- The types of subsequent exchanges are **CREATE_CHILD_SA** (which creates a Child SA) and **INFORMATIONAL** (which deletes an SA, reports error conditions, or does other housekeeping).

- An IKE message flow always consists of a request followed by a response. It is the responsibility of the requester to ensure reliability. If the response is not received within a timeout interval, the requester needs to retransmit the request (or abandon the connection). Every request requires a response.

- An **INFORMATIONAL** request with no payloads (other than the empty Encrypted payload required by the syntax) is commonly used as a **check for liveness**. These subsequent exchanges cannot be used until the initial exchanges have completed.

Fase 1.1 (mensagens iniciais): IKE_SA_INIT

- Mensagem inicial:

HDR, SAi1, KEi, Ni -->

  - O **HDR** (IKE *header*) inclui os SPI, número de versão e várias *flags.*
  - O **SAi1** inclui os algoritmos criptográficos que o *initiator* suporta/propõe para o IKE.
  - O **KEi** indica o valor DH.
  - O **Ni** é o *nonce* do *initiator.*

Mensagens iniciais: IKE_SA_INIT

- Resposta do *responder*:

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

  - Indica a sua preferência no que respeita aos algoritmos criptográficos através do **SAr1**, termina o DH através do **KEr**, envia um *nonce* **Nr** e pede o certificado do *initiator* (opcional).

  - A partir deste ponto da negociação torna-se possível gerar a chave **SKEYSEED** a partir dos valores **KEi** e **KEr** trocados (DH).

  - As chaves seguintes para o **IKE SA** serão geradas a partir da **SKEYSEED**.

Generating Keying Material for the IKE SA

The keys used for the encryption and integrity protection are derived from **SKEYSEED** and are known as **SK_e** (encryption) and **SK_a** (authentication, a.k.a. integrity protection). A separate SK_e and SK_a is computed for each direction.

In addition to the keys SK_e and SK_a derived from the Diffie-Hellman value for protection of the IKE SA, another quantity **SK_d** is derived and used for derivation of further keying material for **Child SAs**.

The notation **SK { ... }** indicates that these payloads are encrypted and integrity protected using that direction's SK_e and SK_a.

Generating Keying Material for the IKE SA

As primeiras chaves geradas a partir da **SKEYSEED** são usadas para confidencialidade e protecção de integridade/autenticação designando-se por **SK_e** (cifra) e **SK_a** (autenticação). Um par para cada direcção.

$$\mathbf{SKEYSEED = prf(Ni \mid Nr, g^{ir}))}$$

[**Nota: g^{ir}** é o resultado do Diffie-Helman (DH) usando os **KEi** e **KEr** trocados entre I e R]

Os *nonces* acrescentam "frescura" à nova chave.

É gerada também a chave **SK_d** que servirá para gerar as chaves para os ***child* SAs**.

$$\mathbf{\{SK\_d \mid SK\_ai \mid SK\_ar \mid SK\_ei \mid SK\_er \mid SK\_pi \mid SK\_pr\} =}$$

$$\mathbf{prf+(SKEYSEED, N_i \mid N_r \mid SPIi \mid SPIr)}$$

A notação SK{ … } indica que o conteúdo está protegido pelas chaves **SK_e** (cifra) e **SK_a** (autenticação) dessa direcção.

As mensagens seguintes da fase 1.2 serão protegidas por *SK_ai, SK_ar , SK_ei e SK_er.*

Generating Keying Material for the IKE SA

g^ir is the shared secret from the ephemeral Diffie-Hellman exchange. g^ir is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus. Ni and Nr are the nonces, stripped of any headers.

The two directions of traffic flow use different keys. The keys used to protect messages from the original initiator are SK_ai and SK_ei. The keys used to protect messages in the other direction are SK_ar and SK_er.

Fase 1.2 (troca inicial): IKE_AUTH

HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi, TSr} -->

- Apenas nesta mensagem é que o *initiator* revela a sua identidade de forma protegida. Prova o conhecimento dos segredos correspondentes ao IDi.
- O campo AUTH protege a integridade da anterior mensagem (a primeira): IKE_SA_INIT.
- Envia o seu certificado CERT (opcional). Se for incluído uma carga CERT, o primeiro certificado DEVE incluir a chave pública usada para verificar o campo AUTH.
- Fornece indicação sobre os certificados de quem assinou o seu certificado digital e pede o do *responder* (CERTREQ).
- O IDr (opcional) pode indicar com qual das identidades do *responder* o *initiator* pretende estabelecer ligação.
- O *initiator* começa a negociar o *child* SA usando o SAi2.
- Propõe os selectores de tráfego TSi, TSr.

Fase 1.2 (troca inicial): IKE_AUTH

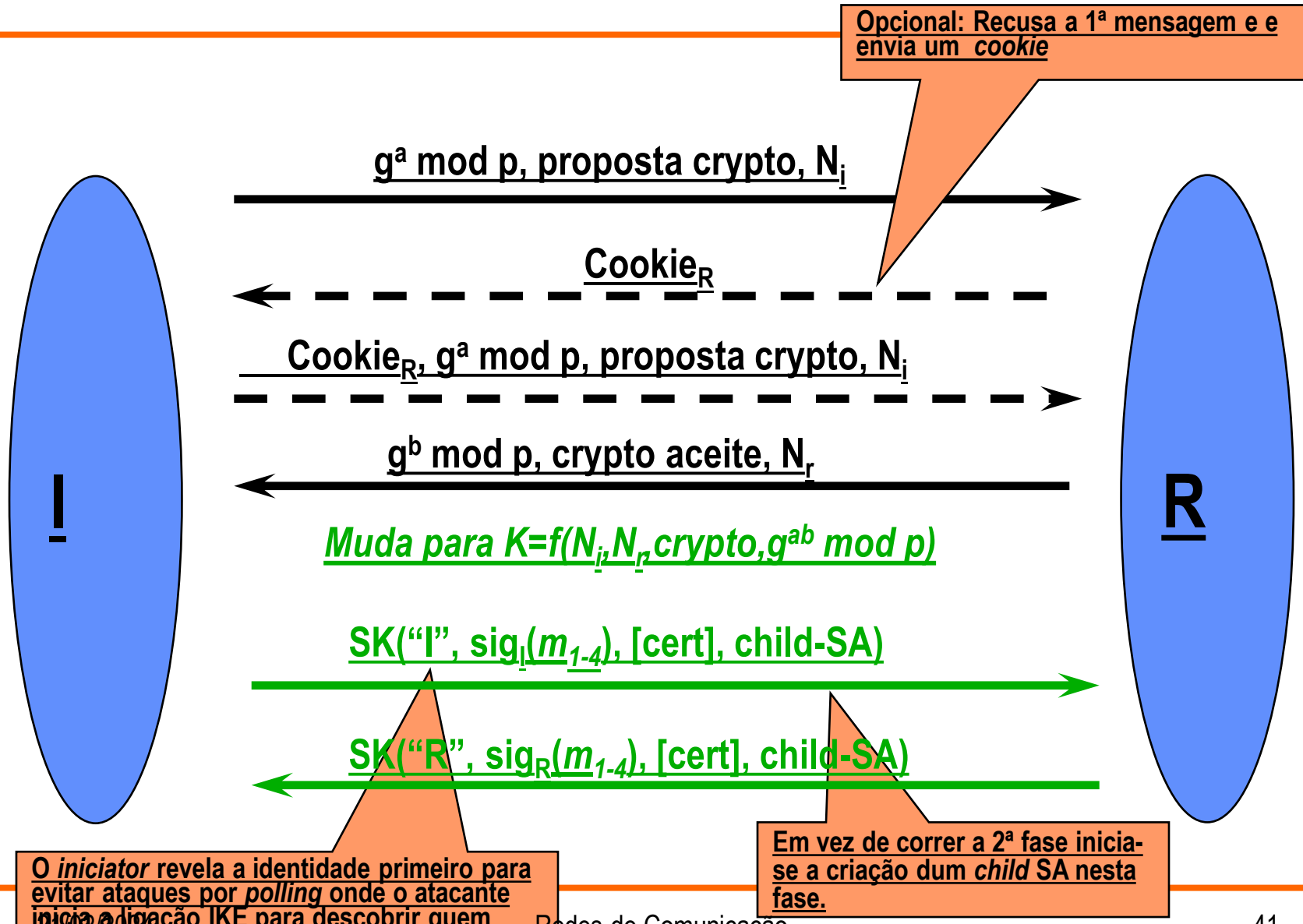
<-- HDR, SK {IDr, [CERT,] AUTH, SAr2, TSi, TSr}

- O *responder* confirma a sua identidade com o IDr, envia o seu certificado CERT, autentica a mensagem anterior usando o campo AUTH e indica qual a *suite* criptográfica que prefere (SAr2) e indica quais os selectores de tráfego a usar, completando a negociação do *child* SA.

Autenticação mútua com o AUTH em IKE_AUTH

- Dois métodos de autenticação
  - **Assinatura digital**
    - Requer certificado(s) individual [CERT]
    - [CERTREQ] indica autenticação por certificados
    - *Initiator* assina a primeira mensagem IKE_SA_INIT concatenada com Nr e prf(SK_pi, IDi'). O Nr e o prf(SK_pi, IDi') não são enviados).
    - *Responder* assina, usando a sua chave privada, a sua mensagem IKE_SA_INIT concatenada com o Ni e prf(SK_pr, IDr'). O Ni e o prf(SK_pr, IDr') não são enviados).
    - [CERT] tem de incluir a chave pública que permita verificar a integridade e autenticação de AUTH.
  - **Chave pré-partilhada** (*Pre-shared Key* (SK_pi, SK_pr))
    - HMAC usando a função prf negociada
    - AUTH = prf(prf(Chave pré-partilhada, "Key Pad for IKEv2"), <octetos da mensagem>)

IKEv2: Fase 1


Opcional: Recusa a 1ª mensagem e e envia um *cookie*
g^a mod p, proposta crypto, N_i
$Cookie_R$
$Cookie_R$, g^a mod p, proposta crypto, N_i
g^b mod p, crypto aceite, N_r
I
R
Muda para K=f(N_i,N_r,crypto,g^{ab} mod p)
SK("I", $sig_I(m_{1-4})$, [cert], child-SA)
SK("R", $sig_R(m_{1-4})$, [cert], child-SA)
O *iniciator* revela a identidade primeiro para evitar ataques por *polling* onde o atacante inicia a ligação IKE para descobrir quem vive no endereço IP
Em vez de correr a 2ª fase inicia-se a criação dum *child* SA nesta fase.

Negociação dos *child* SA no IKE_AUTH

- Estabelecimento do *child* SA em *piggyback* na mensagem IKE_AUTH.

- O *initiator* propõe SAi2 na mensagem 3
- O *responder* responde na mensagem 4

- Tráfego protegido pelo SA é negociado através dos TSi e TSr.
- Qualquer um dos lados pode iniciar a troca de mensagens CREAT_CHILD_SA

Negociation of the *child* SA

The **CREATE_CHILD_SA** request MAY optionally contain a **KE** payload for an additional Diffie-Hellman exchange to enable stronger guarantees of forward secrecy for the Child SA.

The keying material for the Child SA is a function of **SK_d** established during the establishment of the IKE SA, the nonces exchanged during the CREATE_CHILD_SA exchange, and the Diffie-Hellman value (if KE payloads are included in the CREATE_CHILD_SA exchange).

Criar um novo SA: CREATE_CHILD_SA

HDR, SK {SA, Ni, [KEi,] TSi, TSr} -->

- O N só é transportado se se pretender criar novas chaves para o IKE SA.
- Os KE só são enviados se se pretender recalcular o DH (*perfect forward secrecy*)
- Os selectores de tráfego (TSi, TSr) permitem indicar qual o tráfego visado por esta *child* SA.

<-- HDR, SK {SA, Nr, [KEr], TSi, TSr}

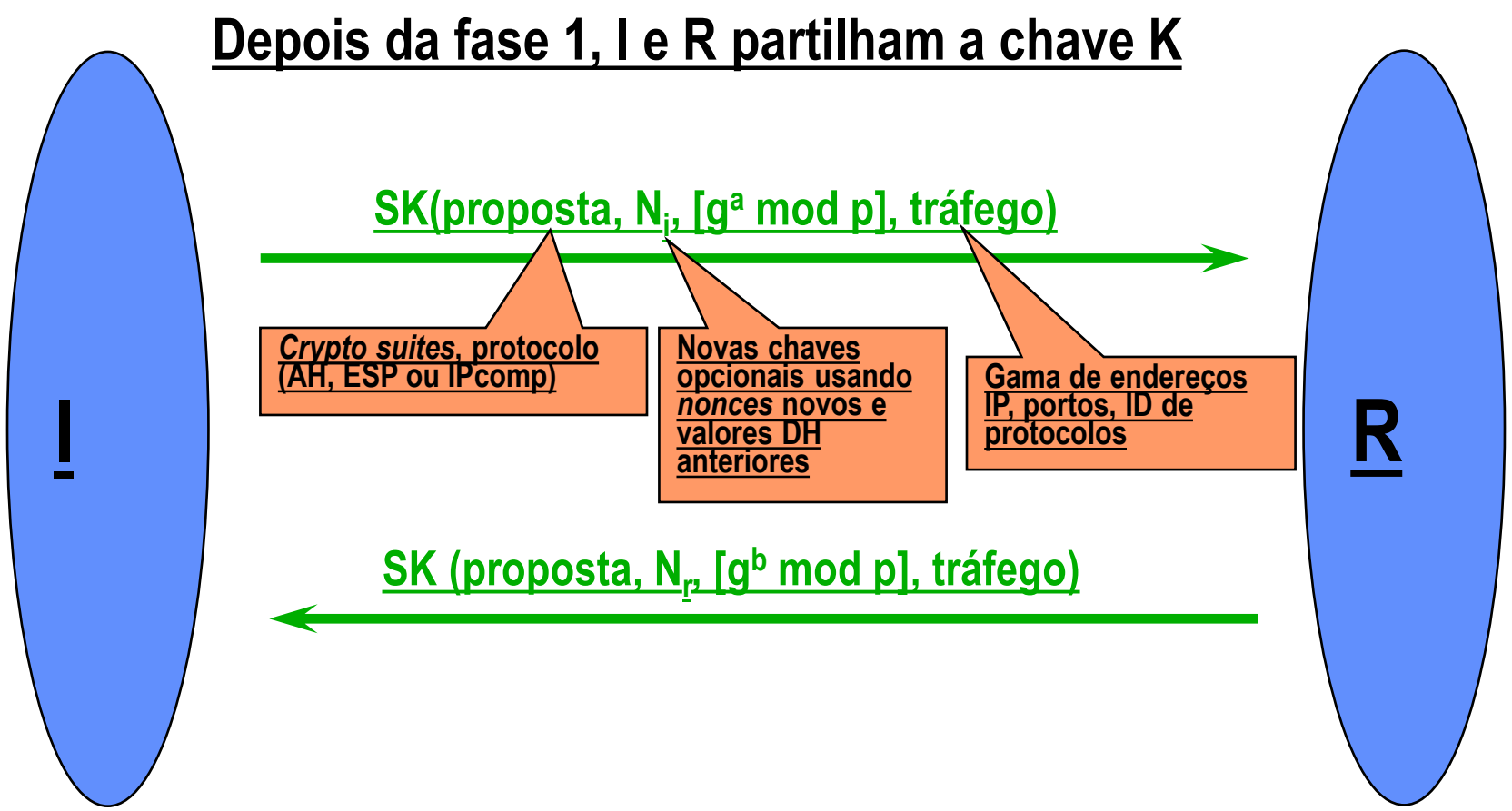
Creation of a new SA: CREATE_CHILD_SA

The **USE_TRANSPORT_MODE** notification MAY be included in a request message that also includes an SA payload requesting a Child SA. It requests that the Child SA use transport mode rather than tunnel mode for the SA created.

If the request is accepted, the response MUST also include a notification of type USE_TRANSPORT_MODE. If the responder declines the request, the Child SA will be established in tunnel mode. If this is unacceptable to the initiator, the initiator MUST delete the SA.

Note: Except when using this option to negotiate transport mode, **all Child SAs will use tunnel mode**.

IKEv2: Fase 2 (Criar um *child* SA)



Pode correr esta fase muitas vezes para criar múltiplos *child* SAs

Chaves para o AH e ESP

- Depois de CREATE_CHILD_SA, a(s) chave(s) para o AH e/ou ESP são geradas a partir de KEYMAT:

KEYMAT = prf+(SK_d, Ni | Nr)

Ni e Nr são novos *nonces* da fase 2

- Para um PFS mais forte:

KEYMAT = prf+(SK_d, g^{ir} (novo) | Ni | Nr)

- Onde g^{ir} são os novos valores DH da fase 2, SK_d foi criado na fase 1, Ni e Nr são novos na fase 2.
- O prf+ de 160 bit é usado duas vezes para gerar a chave de 256 bit para o AES.

Negociação dos algoritmos criptográficos

- O *payload* tipo SA consiste em uma ou mais propostas:
  - Protocolos: IKE, ESP, AH, ESP+AH
  - Algoritmos criptográficos associados com cada protocolo.

  - O *responder* responde com uma escolha baseada na proposta do *initiator*.

Rechaveamento

- As chaves secretas do IKE, ESP e AH devem ser utilizados por tempo limitado.
- Depois do tempo de vida do SA terminar, é necessário criar novas chaves.
- Cada lado é independente para criar novos SA e renegociar as chaves. Os tempos são independentes entre *initiator* e *responder*.
- Depois do novo SA ser negociado apaga-se o anterior.

Rekeying IKE SAs with the CREATE_CHILD_SA Exchange

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

The initiator sends SA offer(s) in the **SA** payload, a nonce in the **Ni** payload, and a Diffie-Hellman value in the **KEi** payload. The KEi payload MUST be included. A new initiator SPI is supplied in the SPI field of the SA payload. Once a peer receives a request to rekey an IKE SA or sends a request to rekey an IKE SA, it SHOULD NOT start any new CREATE_CHILD_SA exchanges on the IKE SA that is being rekeyed. The CREATE_CHILD_SA response for rekeying an IKE SA is:

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

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and a Diffie-Hellman value in the KEr payload if the selected cryptographic suite includes that group. A new responder SPI is supplied in the SPI field of the SA payload.

Rekeying Child SAs with the CREATE_CHILD_SA Exchange

HDR, SK {N(REKEY_SA), SA, Ni, [KEi,], TSi, TSr} -->

The initiator sends SA offer(s) in the **SA** payload, a nonce in the **Ni** payload, optionally a Diffie-Hellman value in the **KEi** payload, and the proposed Traffic Selectors for the proposed Child SA in the **TSi** and **TSr** payloads.
Other notifications may also be sent in a rekeying exchange. Usually, these will be the same notifications that were used in the original exchange; for example, when rekeying a transport mode SA, the USE_TRANSPORT_MODE notification will be used.

The REKEY_SA notification MUST be included in a CREATE_CHILD_SA exchange if the purpose of the exchange is to replace an existing ESP or AH SA. The SA being rekeyed is identified by the SPI field in the Notify payload; this is the SPI the exchange initiator would expect in inbound ESP or AH packets. There is no data associated with this Notify message type. The Protocol ID field of the REKEY_SA notification is set to match the protocol of the SA we are rekeying, for example, 3 for ESP and 2 for AH.

Rekeying Child SAs with the CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA response for rekeying a Child SA is:

<-- HDR, SK {SA, Nr, [KEr,], TSi, TSr}

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and a Diffie-Hellman value in the KEr payload if KEi was included in the request and the selected cryptographic suite includes that group.

The Traffic Selectors for traffic to be sent on that SA are specified in the TS payloads in the response, which may be a subset of what the initiator of the Child SA proposed.

Transporte das mensagens do IKE

- Utiliza:
  - **UDP**
  - **Porto 500**
- Pode utilizar, para compatibilização com o NAT, uma formatação diferente:
  - UDP
  - Porto 4500

- Utiliza *timers* para retransmissão e assim garantir alguma fiabilidade. Funciona num modo semelhante ao **send-and-wait**.

- O **Message ID** (32 bit) permite relacionar pedidos com respostas.

Transport of IKE messages

IKE normally listens and sends on UDP port 500, though IKE messages may also be received on UDP port 4500 with a slightly different format.

Since UDP is a datagram (unreliable) protocol, IKE includes in its definition recovery from transmission errors, including packet loss, packet replay, and packet forgery. IKE is designed to function so long as (1) at least one of a series of retransmitted packets reaches its destination before timing out; and (2) the channel is not so full of forged and replayed packets so as to exhaust the network or CPU capacities of either endpoint. Even in the absence of those minimum performance requirements, IKE is designed to fail cleanly (as though the network were broken).

Transport of IKE messages

Although IKEv2 messages are intended to be short, they contain structures with no hard upper bound on size (in particular, digital certificates), and IKEv2 itself does not have a mechanism for fragmenting large messages. IP defines a mechanism for fragmentation of oversized UDP messages, but implementations vary in the maximum message size supported. Furthermore, use of IP fragmentation opens an implementation to denial-of-service (DoS) attacks [DOSUDPPROT].

Finally, some NAT and/or firewall implementations may block IP fragments.

Use of Sequence Numbers for Message ID

Every IKE message contains a Message ID as part of its fixed header. This Message ID is used to match up requests and responses and to identify retransmissions of messages. Retransmission of a message MUST use the same Message ID as the original message.

The Message ID is a 32-bit quantity, which is zero for the IKE_SA_INIT messages (including retries of the message due to responses such as COOKIE and INVALID_KE_PAYLOAD), and incremented for each subsequent exchange. Thus, the first pair of IKE_AUTH messages will have an ID of 1, the second (when EAP is used) will be 2, and so on. The Message ID is reset to zero in the new IKE SA after the IKE SA is rekeyed.

Each endpoint in the IKE Security Association maintains two "current“ Message IDs: the next one to be used for a request it initiates and the next one it expects to see in a request from the other end. These counters increment as requests are generated and received.

Use of Sequence Numbers for Message ID

Responses always contain the same Message ID as the corresponding request. That means that after the initial exchange, each integer n may appear as the Message ID in four distinct messages: the nth request from the original IKE initiator, the corresponding response, the nth request from the original IKE responder, and the corresponding response.

If the two ends make a very different number of requests, the Message IDs in the two directions can be very different. There is no ambiguity in the messages, however, because the Initiator and Response flags in the message header specify which of the four messages a particular one is.

Note that Message IDs are cryptographically protected and provide protection against message replays. In the unlikely event that Message IDs grow too large to fit in 32 bits, the IKE SA MUST be closed or rekeyed.

IKE SA SPIs and Cookies

The **initial two eight-octet (64bit) fields in the header, called the "IKE SPIs"**, are used as a **connection identifier** at the beginning of IKE packets. Each endpoint chooses one of the two SPIs and MUST choose them so as to be unique identifiers of an IKE SA.

An SPI value of zero is special: it indicates that the remote SPI value is not yet known by the sender.

Incoming IKE packets are mapped to an IKE SA only using the packet's SPI, not using (for example) the source IP address of the packet.

Unlike ESP and AH where only the recipient's SPI appears in the header of a message, in IKE the sender's SPI is also sent in every message. Since the SPI chosen by the original initiator of the IKE SA is always sent first, an endpoint with multiple IKE SAs open that wants to find the appropriate IKE SA using the SPI it assigned must look at the Initiator flag in the header to determine whether it assigned the first or the second eight octets.

IKE SA SPIs and Cookies

Two expected **attacks against IKE** are **state and CPU exhaustion**, where the target is flooded with session initiation requests from forged IP addresses. These attacks can be made less effective if a responder uses minimal CPU and commits no state to a SA until it knows the initiator can receive packets at the address from which it claims to be sending them.

IKE SA SPIs and Cookies

Forma de minimizar os ataques por exaustão de recursos dos *responder*.

O *responder* pode obrigar a que sejam utilizados *cookies* se detectar alguma excesso de pedidos de ligação(*half-open IKE SAs*). Deve responder aos *requests* IKE_SA_INIT com um reply contend a notificação COOKIE (1 a 64 octetos de comprimento).

Mensagem inicial rejeitada, seguintes com *cookies*:

HDR(A,0), SAi1, KEi, Ni -->

<-- HDR(A,0), N(COOKIE)

HDR(A,0), N(COOKIE), SAi1, KEi, Ni -->

<-- HDR(A,B), SAr1, KEr, Nr, [CERTREQ]

HDR(A,B), SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi, TSr} -->

<-- HDR(A,B), SK {IDr, [CERT,] AUTH, SAr2, TSi, TSr}

Cookie = <VersionIDofSecret> | *Hash* (Ni | IPi | SPIi | <secret>)

<secret>: Valor aleatório conhecido apenas do *responder* alterado periodicamente

IKE SA SPIs and Cookies

The first two messages do not affect any initiator or responder state except for communicating the cookie. In particular, the message sequence numbers in the first four messages will all be zero and the message sequence numbers in the last two messages will be one. 'A' is the SPI assigned by the initiator, while 'B' is the SPI assigned by the responder.

An IKE implementation can implement its responder cookie generation in such a way as to not require any saved state to recognize its valid cookie when the second IKE_SA_INIT message arrives. The exact algorithms and syntax used to generate cookies do not affect interoperability and hence are not specified here. The following is an example of how an endpoint could use cookies to implement limited DoS protection.

A good way to do this is to set the responder cookie to be:

Cookie = <VersionIDofSecret> | *Hash* (Ni | IPi | SPIi | <secret>)

where <secret> is a randomly generated secret known only to the responder and periodically changed and | indicates concatenation. <VersionIDofSecret> should be changed whenever <secret> is regenerated. The cookie can be recomputed when the IKE_SA_INIT arrives the second time and compared to the cookie in the received message.

IKE SA SPIs and Cookies

If the cookie matches, the responder knows that the cookie was generated since the last change to <secret> and that IPi must be the same as the source address it saw the first time. Incorporating **SPIi** into the calculation ensures that if multiple IKE SAs are being set up in parallel they will all get different cookies (assuming the initiator chooses unique SPIi's). Incorporating **Ni** in the hash ensures that an attacker who sees only message 2 can't successfully forge a message 3. Also, incorporating **SPIi** in the hash prevents an attacker from fetching one cookie from the other end, and then initiating many IKE_SA_INIT exchanges all with different initiator SPIs (and perhaps port numbers) so that the responder thinks that there are a lot of machines behind one NAT box that are all trying to connect.

If a new value for <secret> is chosen while there are connections in the process of being initialized, an IKE_SA_INIT might be returned with other than the current <VersionIDofSecret>. The responder in that case MAY reject the message by sending another response with a new cookie or it MAY keep the old value of <secret> around for a short time and accept cookies computed from either one. The **responder should not accept cookies indefinitely** after <secret> is changed, since that would defeat part of the DoS protection. The responder should change the value of <secret> frequently, especially if under attack.

IKE SA SPIs and Cookies

The initiator should limit the number of cookie exchanges it tries before giving up, possibly using exponential back-off.

An attacker can forge multiple cookie responses to the initiator's IKE_SA_INIT message, and each of those forged cookie replies will cause two packets to be sent: one packet from the initiator to the responder (which will reject those cookies), and one response from responder to initiator that includes the correct cookie.

The Internet Security Association and Key Management Protocol (ISAKMP) [ISAKMP] fixed message header includes two eight-octet fields called "cookies", and that syntax is used by both IKEv1 and IKEv2, although in IKEv2 they are referred to as the "IKE SPI" and there is a new separate field in a Notify payload holding the cookie.

Cryptographic Algorithm Negotiation

The payload type known as "**SA**" indicates a proposal for a set of choices of IPsec protocols (IKE, ESP, or AH) for the SA as well as cryptographic algorithms associated with each protocol.

A SA payload consists of one or more proposals. Each proposal includes one protocol. Each protocol contains one or more transforms -- each specifying a cryptographic algorithm. Each transform contains zero or more attributes (attributes are needed only if the Transform ID does not completely specify the cryptographic algorithm).

Cryptographic Algorithm Negotiation

This hierarchical structure was designed to efficiently encode proposals for cryptographic suites when the number of supported suites is large because multiple values are acceptable for multiple transforms. The responder MUST choose a single suite, which may be any subset of the SA proposal following the rules below.

Each proposal contains one protocol. If a proposal is accepted, the SA response MUST contain the same protocol. The responder MUST accept a single proposal or reject them all and return an error. The error is given in a notification of type NO_PROPOSAL_CHOSEN.

Each IPsec protocol proposal contains one or more transforms. Each transform contains a Transform Type. The accepted cryptographic suite MUST contain exactly one transform of each type included in the proposal. For example: if an ESP proposal includes transforms ENCR_3DES, ENCR_AES w/keysize 128, ENCR_AES w/keysize 256, AUTH_HMAC_MD5, and AUTH_HMAC_SHA, the accepted suite MUST contain one of the ENCR_ transforms and one of the AUTH_ transforms. Thus, **six combinations are acceptable**.

Nonces

- Indica ***n****umber* ***once*** ou ***No****n se****nse*** – depende dos autores consultados.
- Gerados pseudo-aleatóriamente.
- Pelo menos 128 bits e pelo menos metade da dimensão da chave da função PRF negociada.
- Permite criar “frescura” na criação de chaves para os Child SA a partir dos DH anteriores.
- Permite criar novos *child* SA sem novas fase 1 na troca de mensagens.

Resuma da derivação das chaves no IKE

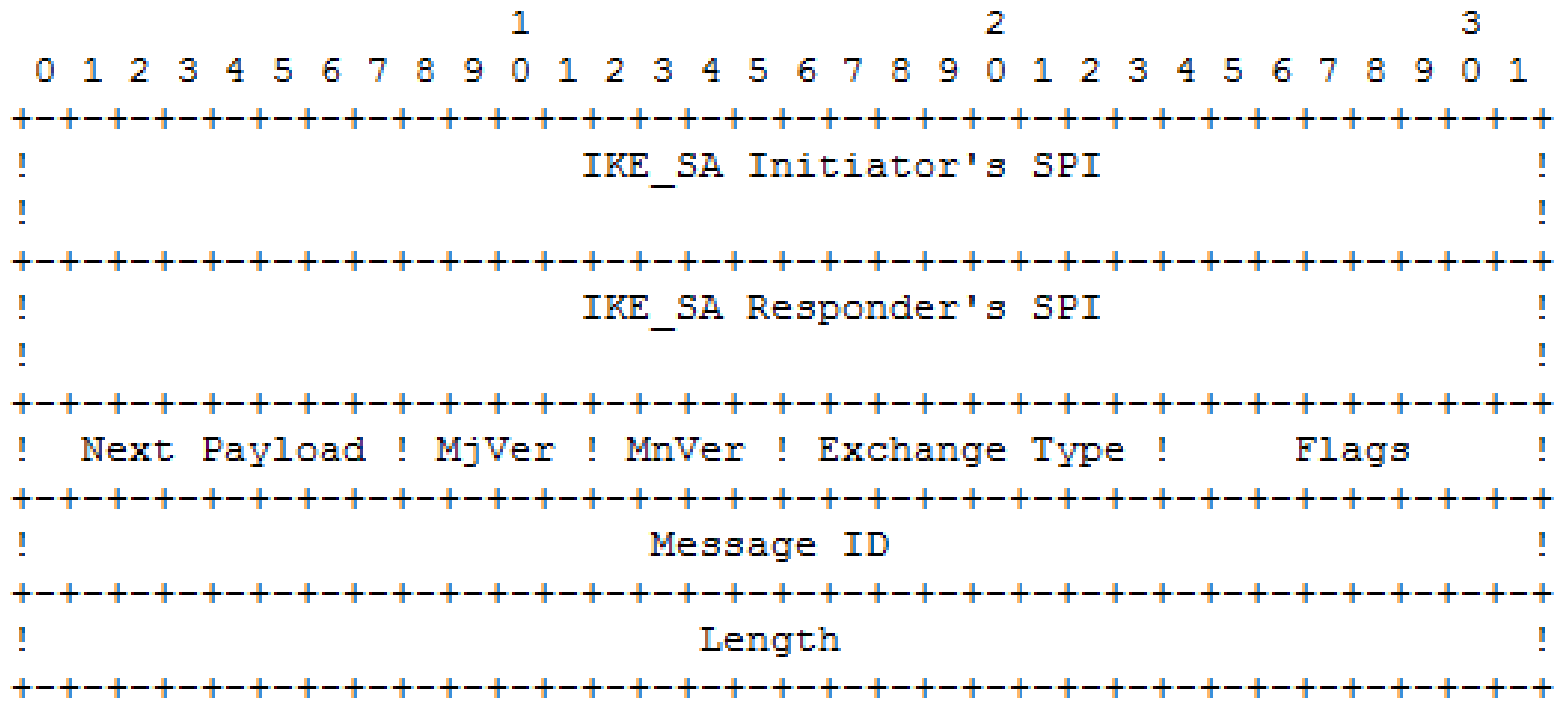
Troca inicial: Valores públicos g^i, g^r e *nonces* N_i e N_r

Deriva a chave mestre:
SKEYSEED = PRF(N_i | N_r, g^{ir})

<u>Deriva outras chaves usando a chave mestre:</u>
{SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr} = prf+(SKEYSEED, N_i | N_r| SPIi|SPIr)

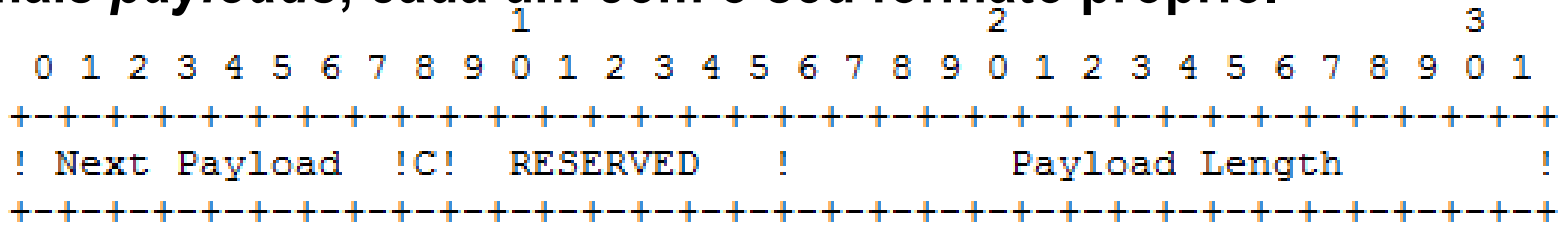
<u>Deriva as chaves para os *child* SA:</u>
KEYMAT=prf+(SK_d, [g^{ir} |] N_i | N_r)

Formato do *header* das mensagens IKE



Header genérico dos *payloads* do IKE

A mensagem IKE é transportada pelo UDP e inclui o *header* IKE e um ou mais *payloads*, cada um com o seu formato próprio.



Valores e tipos do campo Next Payload

Payload 0
RESERVED 1-32
Security Association SA 33
Key Exchange KE 34
Identification - Initiator IDi 35
Identification - Responder IDr 36
Certificate CERT 37
Certificate Request CERTREQ 38
Authentication AUTH 39
Nonce Ni, Nr 40
Notify N 41
Delete D 42
Vendor ID V 43
Traffic Selector - Initiator TSi 44
Traffic Selector - Responder TSr 45
Encrypted E 46
Configuration CP 47
Extensible Authentication EAP 48
RESERVED TO IANA 49-127
PRIVATE USE 128-255

Cisco - Configuration of IPsec VPN with IKEv2 and PSK authentication

To perform this task, we need to configure IPsec main components that include:

- IKEv2 Proposal
- IKEv2 Policy
- IKEv2 Profile
- IKEv2 Keyring
- IPsec transform set
- Crypto Map (the **other option is to define IPSec profile and apply it on a GRE tunnel**)

Example: https://nil.uniza.sk/site-to-site-ikev2-ipsec-vpn-using-pre-shared-key-authentication-simple-configuration-example-on-cisco-routers/

FIM