



Seguridad en SNMP

Gestión de Redes

¿Es seguro SNMP?

- ❑ SNMPv1?
- ❑ SNMPv2?
- ❑ SNMPv3?

Soluciones de seguridad en SNMPv3

- ¿Cómo mejorar SNMP?
 - ▣ Mensajes cifrados y autenticados
- SNMPv3 – USM Estándar para introducir criptografía
- Otras alternativas
 - ▣ SNMPv3 sobre túneles TLS o DTLS
 - ▣ SNMPv3 sobre túneles SSH
 - ▣ SNMPv3 con Kerberos

Práctica

Configuración SNMPv3/USM en NetSNMP

Configurar SNMPv3/USM en Netsnmp

- Parámetros de configuración:

- ▣ **Nombre de usuario**

- ▣ **Métodos/algoritmos:**

- authNopriv (MD5/SHA)

- authPriv(MD5/SHA, DES/AES)

Nota: DES, AES y SHA soportados con OpenSSL

- ▣ **Password** o passwords de autenticación y cifrado

- ▣ **EngineID:** por defecto el sistema asigna un valor, pero se puede definir

snmpd.conf

- Crear usuario:
 - ▣ **createUser** sustituye a **com2sec**
- Tipo de acceso:
 - ▣ **rouser/rwuser** sustituye a **rocommunity/rwcommunity**

```
agentAddress udp:161,udp6:[::1]:161
```

```
createUser admin SHA admin1234 AES
```

```
createUser oper SHA oper1234
```

```
rouser oper
```

```
rwuser admin priv -V network
```

```
view network included .iso.org.dod.internet.mgmt.mib-2.ip  
view network included .iso.org.dod.inernet.mgmt.mib-2.interfaces
```

```
snmpget -v 3 -u oper -l authNoPriv -a SHA -A oper1234 10.0.123.3 SNMPv2-MIB::sysUpTime.0
```

```
snmpget -v 3 -u admin -l authPriv -a SHA -A admin1234 -x AES -X admin1234 10.0.123.3 SNMPv2-MIB::sysUpTime.0
```

Valores por defecto en el gestor

- Editar `$HOME/.snmp/snmp.conf`

```
defSecurityName admin
defContext ""
defAuthType SHA
defPrivType AES
defSecurityLevel authPriv
defAuthPassphrase admin1234
defPrivPassphrase admin1234
defVersion 3
```

- Configuramos el fichero de sólo lectura para el usuario.
- Ahora *snmpget* asume los valores definidos por defecto.

```
snmpget 10.0.123.3 sysUpTime.0
```

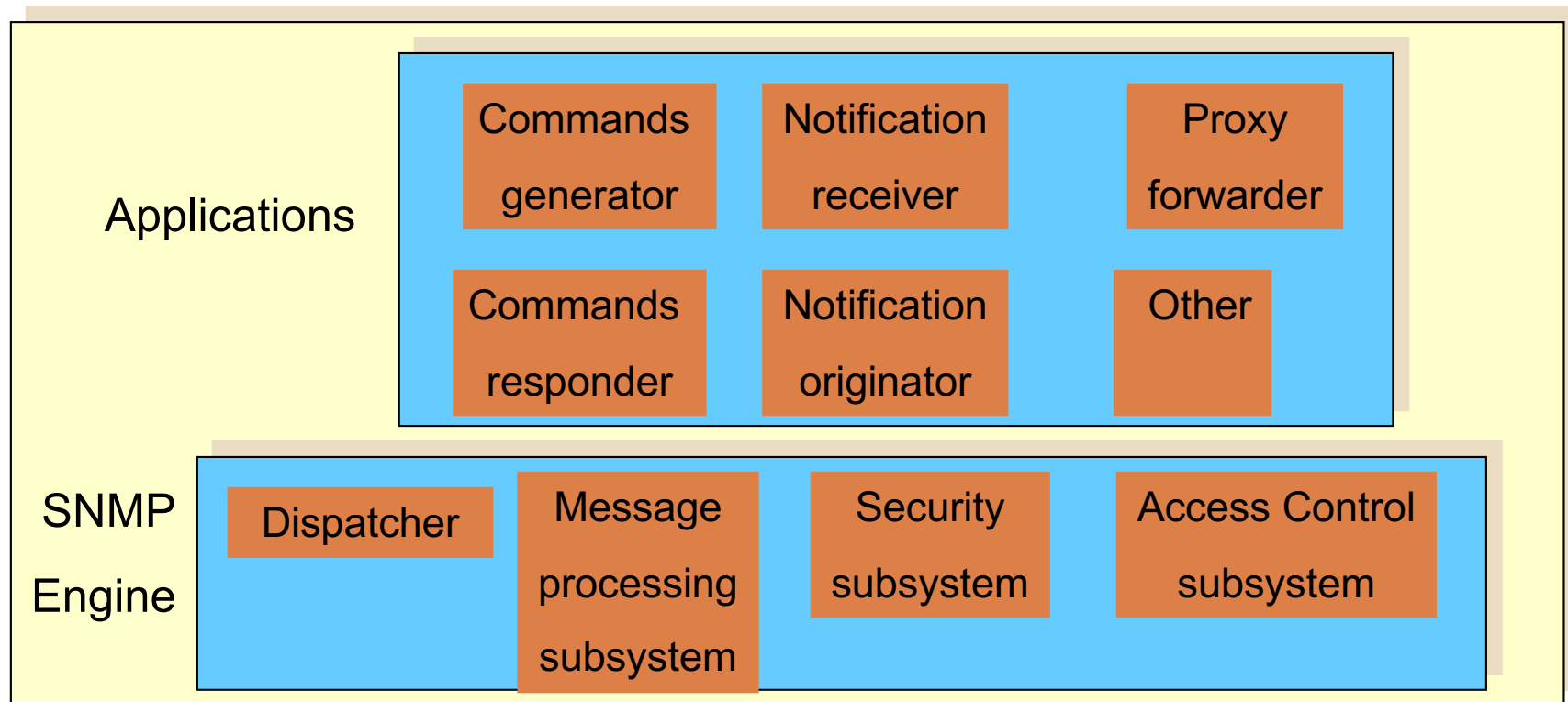
Ahora vamos con la teoría

¿Cómo es por dentro SNMPv3?

Arquitectura SNMPv3

9

- Compuesta por entidades SNMP que interactúan entre ellas:
 - ▣ Cada entidad ofrece una porción de la capacidad SNMP.
 - ▣ Puede ser parte de un agente, gestor o combinación de ambos.
 - ▣ Entidad compuesta por módulos que interactúan para ofrecer un servicio.



- El motor se identifica de forma única por snmpEngineID
- **Dispatcher:**
 - ▣ Soporta varias versiones
 - ▣ Interactúa con:
 - Aplicaciones: aceptar PDUs de las aplicaciones para enviarlas y entregar PDUs de entrada a la aplicación.
 - Subsistema de procesamiento de mensajes: le pasa los mensajes entrantes para extraer las PDUs y recibe los salientes para enviarlos.
 - Red: enviar y recibir mensajes.
- **Subsistema de procesamiento de mensajes:** genera los mensajes a enviar y extrae PDUs de los mensajes recibidos.
- **Subsistema de seguridad:** provee servicios de seguridad (autenticación y privacidad de los mensajes).
- **Subsistema de control de acceso:** provee servicios de autorización para comprobación de acceso.

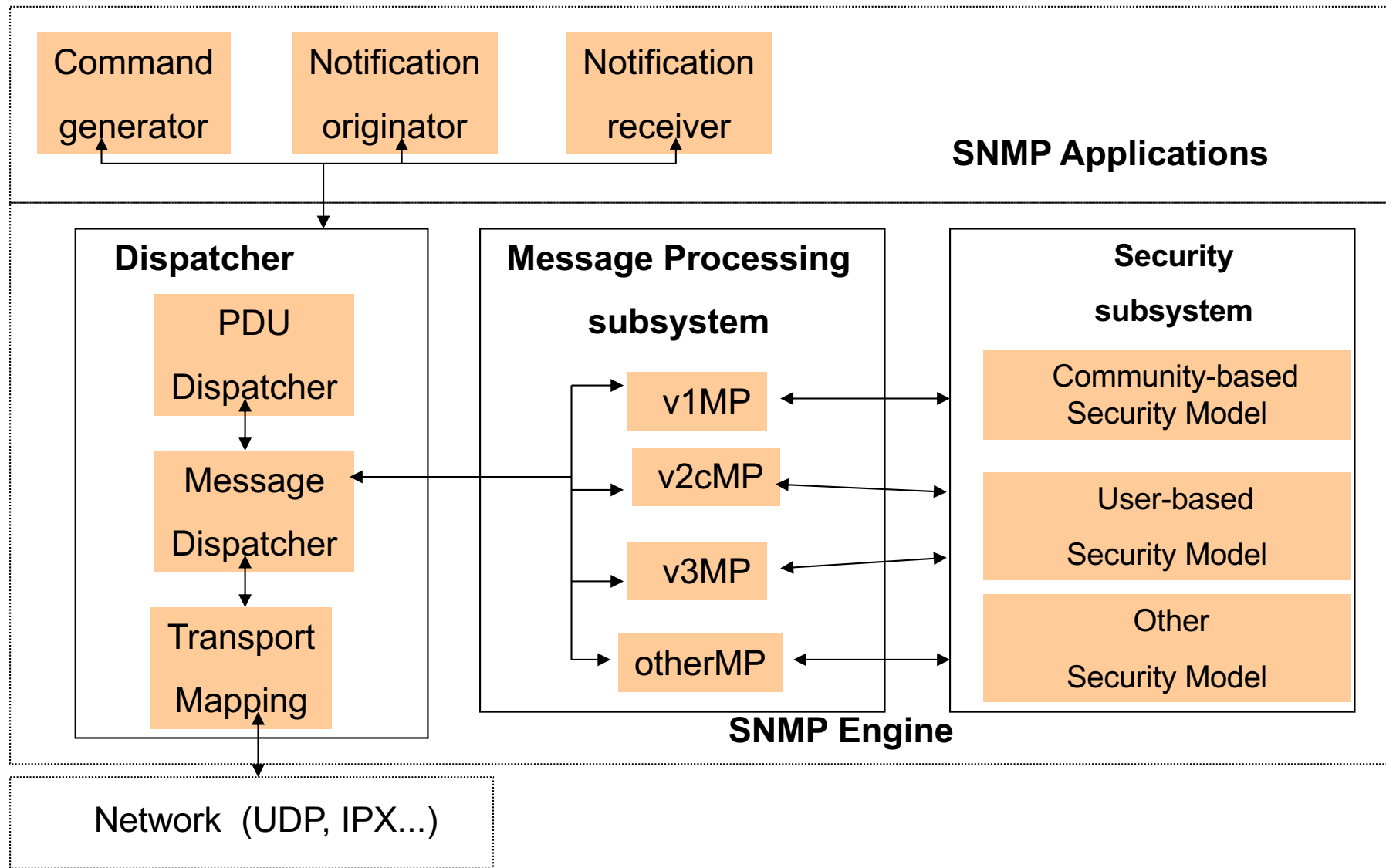
Módulos de una entidad: Aplicaciones

11

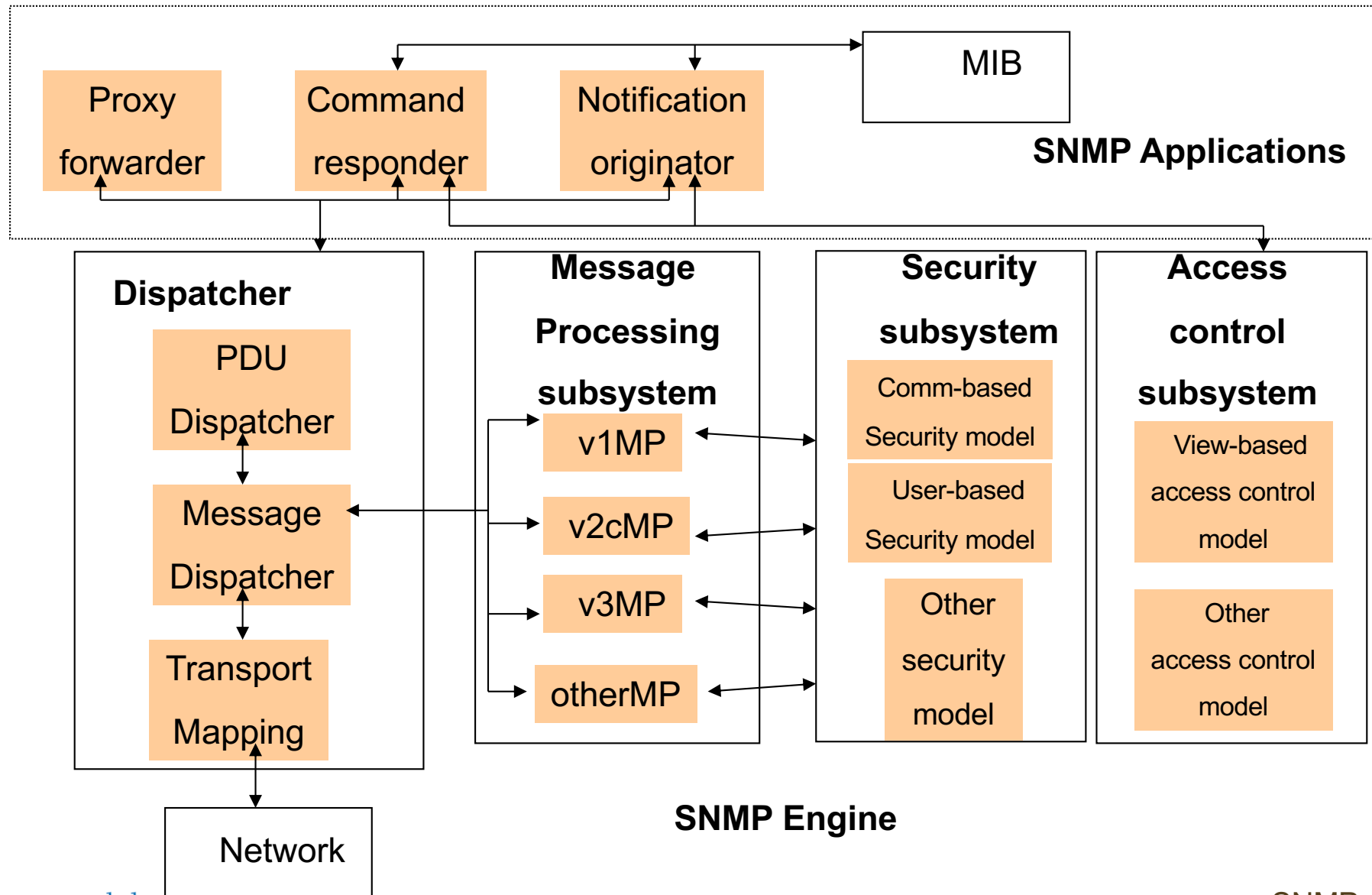
- ❑ **Generador de comandos:** genera PDU de peticiones Get, GetNext, GetBulk y/o Set y procesa las respuestas.
- ❑ **Generador de respuestas de comandos:** contesta a las peticiones Get, GetNext, GetBulk y/o Set de un determinado gestor (snmpEngineID) que tiene acceso (control de acceso).
- ❑ **Generador de notificaciones:** monitoriza eventos o condiciones en un sistema y genera traps o inform.
- ❑ **Receptor de notificaciones:** escucha mensajes de notificación y responde a las peticiones de PDU Inform.
- ❑ **Proxy:** reenvío de mensajes SNMP si actúa como proxy.

Esquema de un Gestor SNMP tradicional

12



Esquema de un Agente SNMP tradicional



Módulos de Aplicaciones

Aplicaciones: procesamiento de PDUs

15

- Procedimiento seguido por cada aplicación cuando genera PDUs o procesa PDUs recibidas.
- Mediante diálogo con el Dispatcher a través de **primitivas**.
- Tipos de aplicaciones y primitivas usadas para dialogar con Dispatcher:
 - ▣ **Command Generator**: sendPdu, processResponsePdu.
 - ▣ **Command Responder**: registerContextEngineID, unregisterContextEngineID, processPdu, returnResponsePdu.
 - ▣ **Notification Generator**: sendPdu, processResponsePdu.
 - ▣ **Notification Receiver**: processPdu, returnResponsePdu.
 - ▣ **Proxy Forwarder** procesa cuatro tipos de PDU:
 - Generación de comandos.
 - Generación de notificación.
 - Respuesta.
 - Informe.

- RFC 2273 define tres MIBs que soportan aplicaciones SNMPv3:
 - ▣ **Management Target.** snmpTargetsObject define:
 - Destinos de notificaciones y proxy: snmpTargetAddrTable.
 - Parámetros usados en envío de mensaje: snmpTargetParamsTable
 - ▣ **Notification.** snmpNotifyObjectsGroup permite configurar notificaciones:
 - snmpNotifyTable: selecciona destino/s de la notificación en la tabla snmpTargetsAddrTable.
 - snmpNotifyFilterTable: define filtros para limitar el nº de notificaciones generadas para un destino particular.
 - snmpNotifyFilterProfileTable: asocia filtros a una entrada en la tabla snmpTargetsAddrTable.
 - ▣ **Proxy.** snmpProxyObjectGroup contiene una tabla con parámetros para configuración remota de un proxy .

Parámetros de primitivas

17

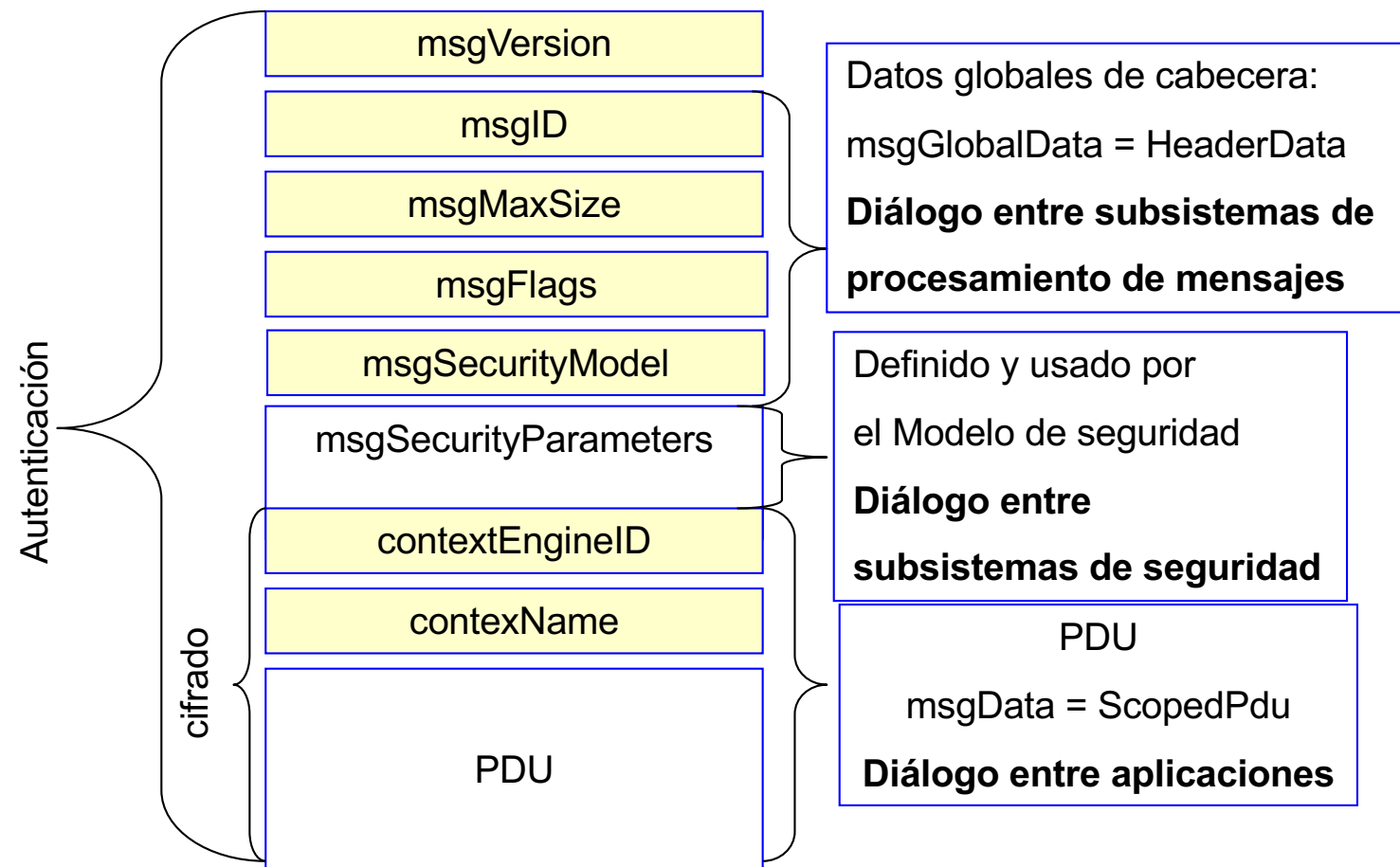
- Se define unas primitivas que se utilizan para establecer el diálogo entre los distintos módulos.
- Se utiliza la siguiente terminología (parámetros de primitivas):
 - ▣ **snmpEngineID**: identifica de forma única un motor SNMP y su entidad.
 - ▣ **contextEngineID**: identifica una entidad SNMP que puede realizar una instancia de un contexto con un ContextName particular. Su valor coincide con el de snmpEngineID.
 - ▣ **contextName**: identifica un contexto particular de un motor SNMP. Un motor SNMP puede manejar varios contextos para la definición del control de acceso.
 - ▣ **scopedPDU**: valores de contextEngineID, contextName y PDU SNMP se pasan al subsistema de seguridad.
 - ▣ **snmpMessageProcessingModel**: SNMPv1, SNMPv2 o SNMPv3.
 - ▣ **snmpSecurityModel**: SNMPv1, SNMPv2c o USM.
 - ▣ **snmpSecurityLevel**: nivel de seguridad para enviar mensajes y procesar operaciones (noAuthnoPriv, authNopriv o authPriv).
 - ▣ **securityName**: string que representa un principal, que identifica la entidad (usuario) que solicita los servicios y procesamientos. Se pasa en las primitivas a Dispatcher, Message Processing, Security y Access Control.

Procesamiento de Mensajes

Subsistema de procesamiento de mensajes

19

- Dialoga con Dispatcher para procesar los mensajes.
- Definición del formato del mensaje en ASN.1 (RFC3412):

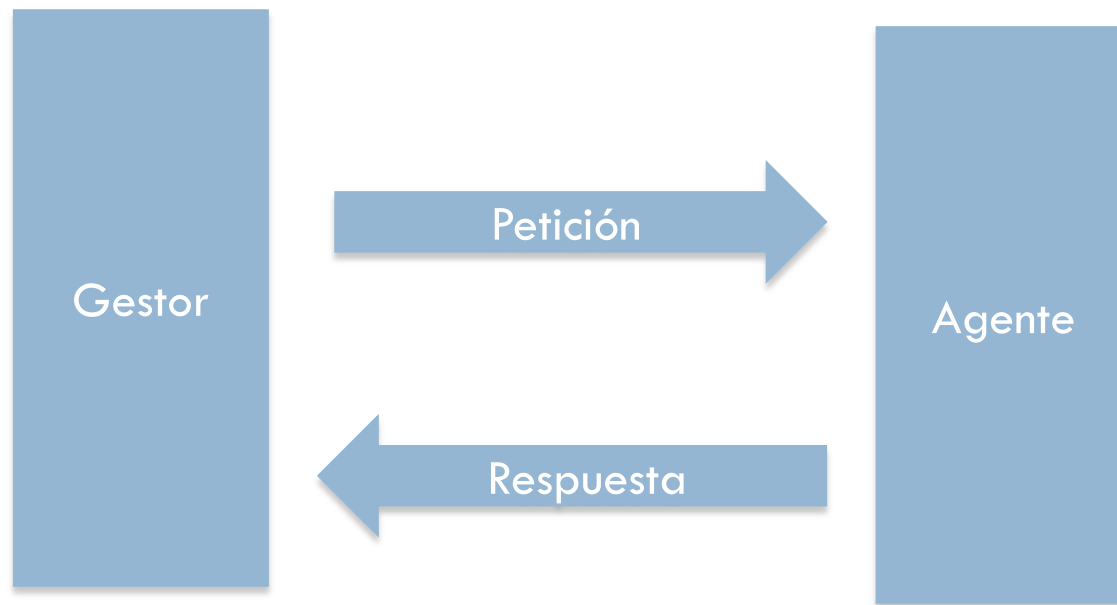


Formato de mensaje SNMPv3

20

- **msgVersion**: versión (3).
- **msgID**: nº único usado para un mensaje y su respuesta.
- **msgMaxSize**: máximo tamaño soportado por el transmisor.
- **msgFlags** contiene tres flags en los últimos tres bits de un octeto:
 - ▣ **reportableFlags**: se debe enviar respuesta (activo en mensajes set, get, Inform), así si falla la decodificación se sabe que requiere respuesta.
 - ▣ **privFlag**: mensaje cifrado para ofrecer privacidad (si authFlag = 1).
 - ▣ **authFlag**: mensaje autenticado.
- **msgSecurityModel**: modelo de seguridad SNMPv1,v2c ó USM (3).
- **msgSecurityParameters**: parámetros intercambiados entre subsistemas de seguridad de las dos entidades.
- **contextEngineID**: ayuda a identificar la aplicación a la que hay que entregar la PDU para procesarla en el receptor. El valor es puesto por la aplicación que genera la PDU.
- **contextName**: identificador de contexto particular.
- **PDU**: PDU con formato SNMPv2.

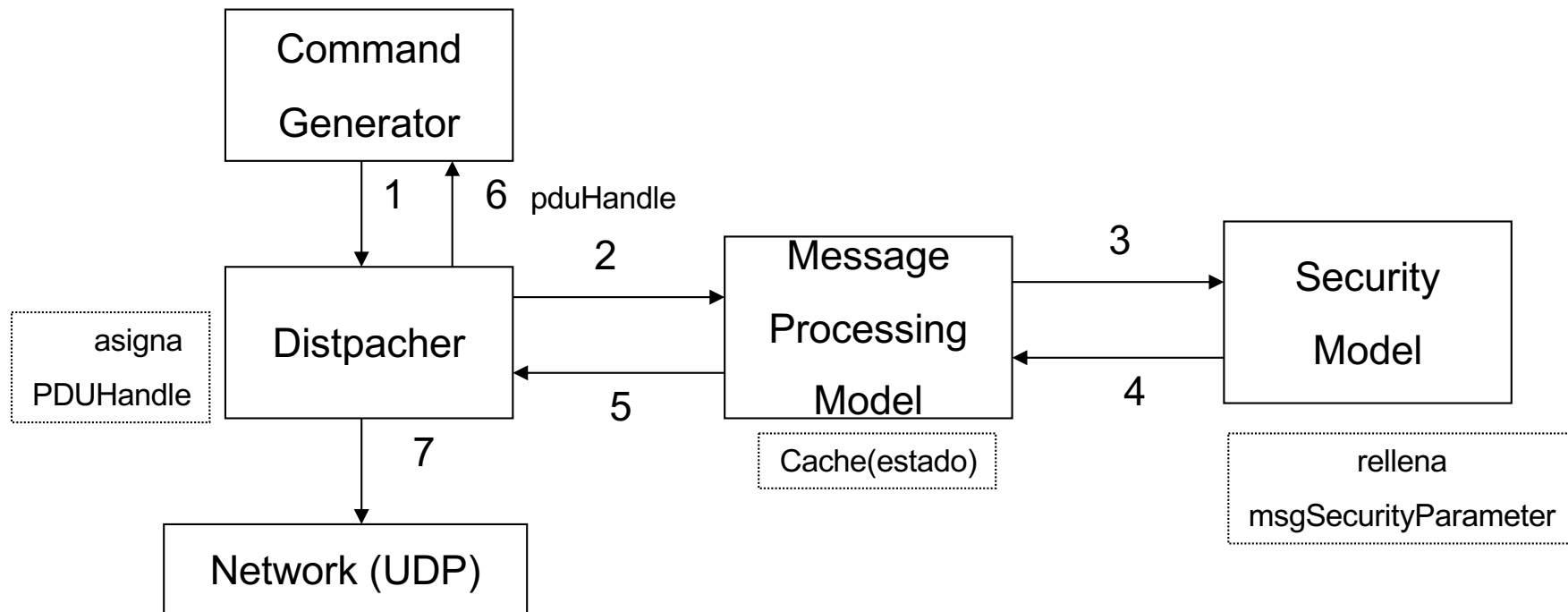
Procesamiento de mensajes Gestor-Agente



Petición en el Gestor

22

Mensaje saliente



Petición en el Gestor

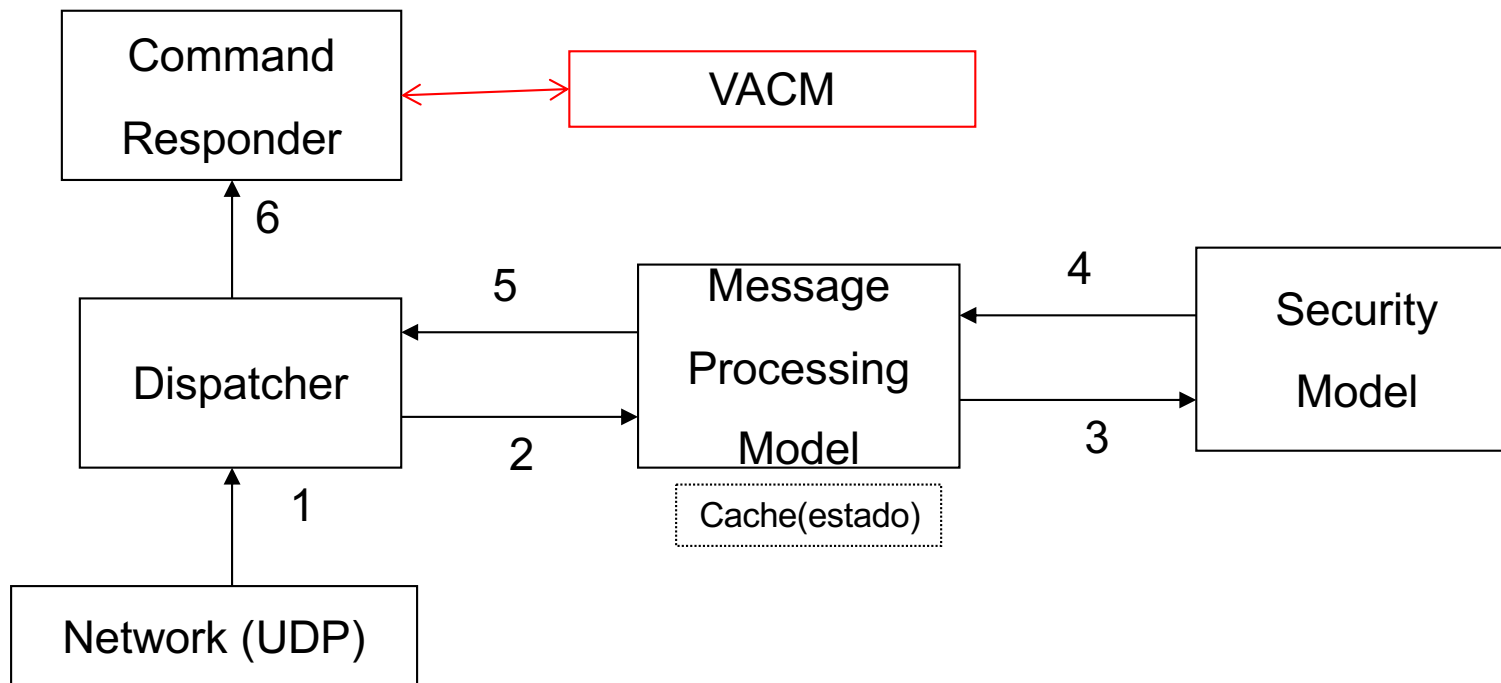
23

- **Command Generator:** selecciona parámetros securityLevel, transportDomain, transportAddress, contextEngineID y contextName.
- **Dispatcher:**
 - ▣ Genera PDUHandle (identificador de PDU) y la pasa a command generator y processing message.
 - ▣ Pasa parámetros y mensaje a message processing.
 - ▣ Envía el mensaje procesado a la red.
- **Message processing:**
 - ▣ Prepara el mensaje.
 - ▣ Lo envía a Security Model para que añada parámetros de seguridad.
 - ▣ Devuelve el mensaje completo a Dispatcher para que lo envíe.
 - ▣ Mantiene estado en cache: msgID, PDUHandle, securityModel, securityName, securityLevel, contextEngineID y contextName.
- **Security Model:** rellena parámetros de seguridad.

Petición en el Agente

24

Mensaje entrante

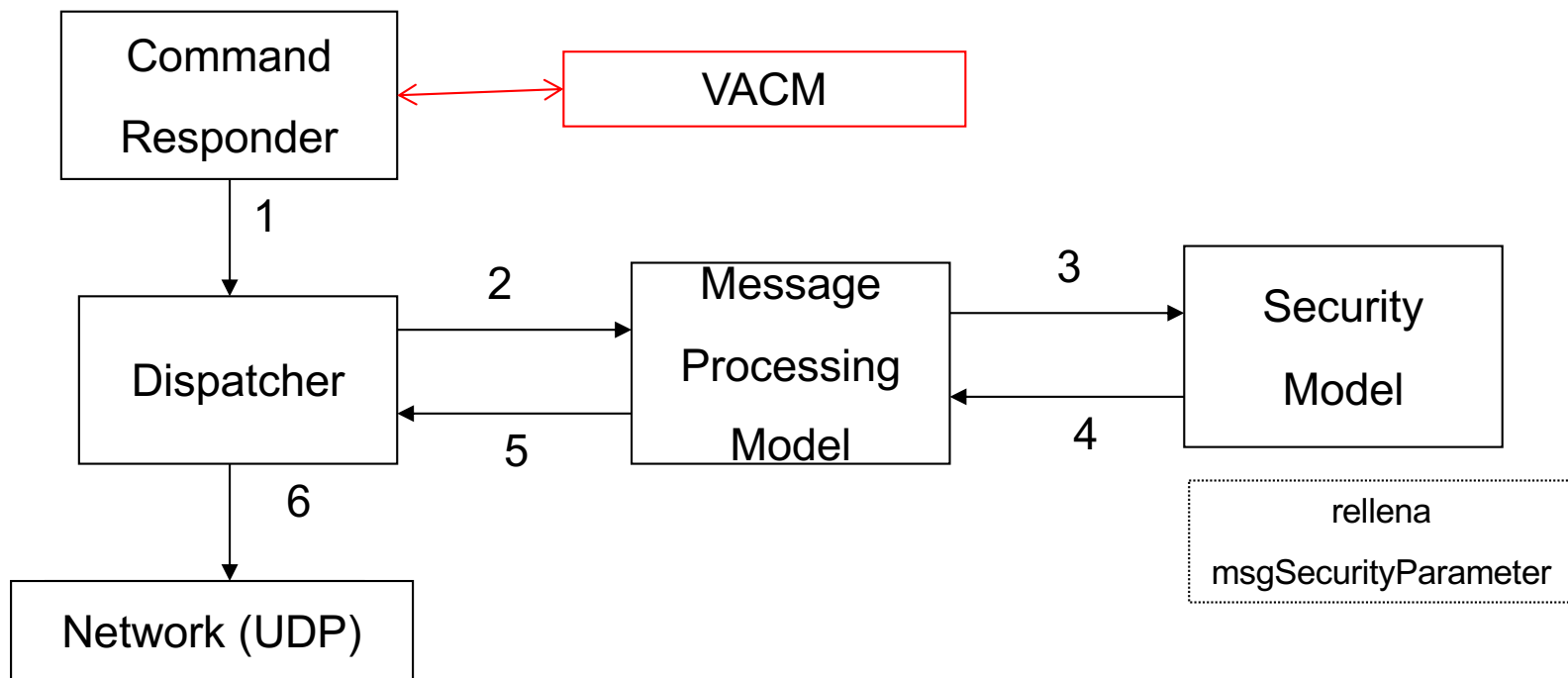


- **Dispatcher:**
 - ▣ Cuando recibe mensaje de la red lo entrega a message processing.
 - ▣ Cuando message processing le devuelve la PDU identifica la aplicación y lo entrega al módulo correspondiente (Command Responder).
- **Message processing:**
 - ▣ Comprueba modelo de seguridad y lo entrega a security model.
 - ▣ Cuando recibe respuesta del mensaje del security model extrae la PDU y la envía a Dispatcher guardando en cache información de estado (versión, ID, MaxSize, seguridad, puerto....).
- **Security Model:** procesa el mensaje (descifra, comprueba autenticidad) de acuerdo con los parámetros de seguridad y lo devuelve a message processing.
- **Command Responder:** recibe la PDU y la procesa para responder.

Respuesta en el Agente

26

Mensaje saliente



Respuesta en el Agente

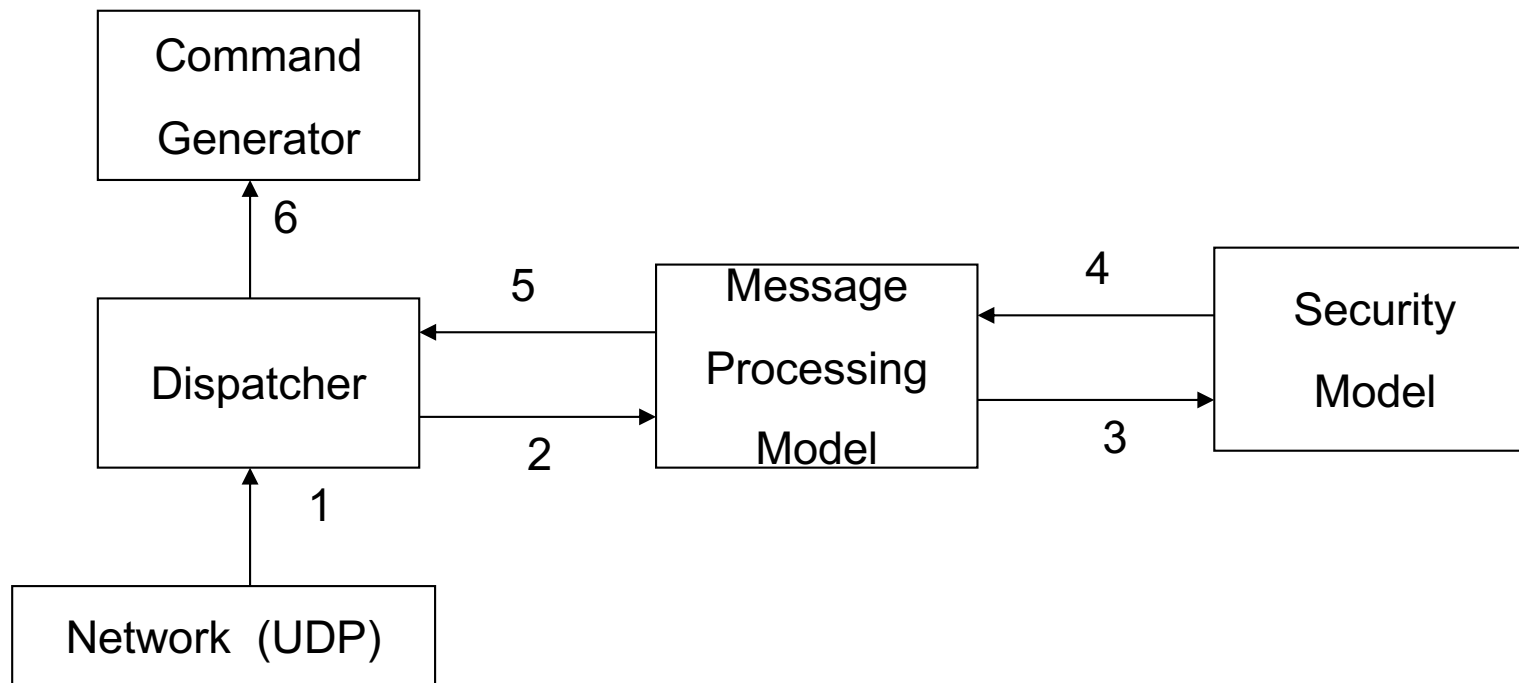
27

- **Command Responder:** genera PDU de respuesta y la envía a Dispatcher.
- **Dispatcher:**
 - ▣ Envía la PDU a processing message para que genere el mensaje de respuesta.
 - ▣ Cuando recibe el mensaje de processing message lo envía a por la red (lo entrega a UDP).
- **Message processing:**
 - ▣ Prepara el mensaje.
 - ▣ Lo envía a Security Model para que añada parámetros de seguridad.
 - ▣ Devuelve el mensaje completo a Distpacher para que lo envíe.
- **Security Model:** rellena parámetros de seguridad y lo devuelve a message processing.

Respuesta en el Gestor

28

Mensaje entrante



- **Dispatcher:**
 - ▣ Cuando recibe mensaje de la red lo entrega a message processing.
 - ▣ Cuando message processing le devuelve la PDU identifica la aplicación y lo entrega al módulo correspondiente.
- **Message processing:**
 - ▣ Comprueba que los parámetros del mensaje coinciden con los de la cache.
 - ▣ Pasa el mensaje al security model y cuando lo recibe, extrae la PDU y la envía a Dispatcher.
- **Security Model:** procesa el mensaje (descifra, comprueba autenticidad) de acuerdo con los parámetros de seguridad y lo devuelve a message processing.
- **Command Generator:** recibe la PDU de respuesta y la procesa.

Modelo de seguridad USM

Modelo de seguridad USM

31

- Modelo de seguridad basado en usuario definido en RFC3414:
- Trata los siguientes temas:
 - ▣ **Autenticación:**
 - Integridad de datos y autenticación de origen.
 - Utiliza el código de autenticación de mensaje HMAC con las funciones hash MD5 o SHA-1.
 - ▣ **Temporización:** protección contra réplica o retardo de mensajes.
 - ▣ **Privacidad:**
 - Protege contra revelación del contenido del mensaje.
 - Utiliza el algoritmo DES en modo CBC (cipher block chaining).
 - ▣ **Formato del mensaje:** define el formato de msgSecurityParameters para soportar privacidad, autenticación y temporización.
 - ▣ **Descubrimiento:** define procedimiento por el cual un motor SNMP obtiene información sobre otro.
 - ▣ **Gestión de claves:** define procedimientos para generar, usar y actualizar claves.

Motor autoritario

32

- Definido en RFC2274. Declarado en ASN.1.
- Se denomina motor SNMP autoritario al que es:
 - ▣ Receptor de mensajes que esperan respuesta: Get, GetNext, GetBulk, Set o Info.
 - ▣ Transmisor de mensajes que no esperan respuesta: SNMPv2-Trap, Response o Report.
- Funciones del motor SNMP autoritario:
 - ▣ Mantenimiento del reloj:
 - Incluye marcas de tiempo en los mensajes para que los no autoritarios sincronicen sus relojes.
 - Los no autoritarios incluyen estimaciones del tiempo actual en el destino (autoritario) para que este lo valore.
 - ▣ Proceso de localización de clave: contiene clave del principal guardada en el motor autoritario.

- Estructura de **msgSecurityParameters**:
 - ▣ **msgAuthoritativeEngineID**: snmpEngineID del motor autoritario.
 - ▣ **msgAuthoritativeEngineBoots**: snmpEngineBoots del motor autoritario. Es decir, el número de veces que se ha (re)iniciado desde su configuración inicial.
 - ▣ **msgAuthoritativeEngineTime**: snmpEngineTime del motor autoritario. Representa los segundos que han pasado desde la última vez que se incrementó snmpEngineBoots. El motor autoritario incrementa su valor cada segundo. El motor no autoritario mantiene una estimación del valor para cada motor autoritario.
 - ▣ **msgUserName**: el principal en nombre del cual se intercambia el mensaje.
 - ▣ **msgAuthenticationsParameters**: Información de autenticación. Código HMAC para el caso de USM. NULL si no hay autenticación.
 - ▣ **msgPrivacyParameters**: Información de cifrado. Para USM contiene IV, el valor del vector de inicialización de DES CBC. NULL si no se usa

Mecanismo de temporización

34

- Gestión de reloj.
 - ▣ Motores autoritarios mantienen snmpEngineBoots y snmpEngineTime.
 - ▣ Si snmpEngineTime alcanza el máximo valor ($2^{31}-1$) se incrementa snmpEngineBoots.
- Sincronización.
 - ▣ Un motor en el papel de no autoritario mantienen una copia por cada motor autoritario conocido de:
 - snmpEngineBoots del motor autoritario.
 - snmpEngineTime: estimación local del tiempo en el autoritario.
 - latestReceivedEngineTime: último tiempo recibido.
 - ▣ Cuando un motor no autoritario recibe un mensaje auténtico:
 - Lo acepta si el tiempo del mensaje (boots+time) es mayor que el último recibido (no réplica).
 - Si un mensaje es aceptado actualiza Boots, Time y latestTime.
 - Si un mensaje llega desordenado, es rechazado.

Mecanismo de temporización

35

- Varios factores provocan que los relojes agentes y gestor tengan ligeras diferencias:
 - ▣ Retardo (RTT) de mensajes
 - ▣ Diferentes frecuencias de relojes
- Se aceptan mensajes dentro de una determinada ventana de tiempo (150 seg).

Comprobación de tiempo en receptores autoritarios

36

- Compara snmpEngineBoots y snmpEngineTime con los valores en el mensaje recibido (msgAuthoritativeEngineBoots y msgAuthoritativeEngineTime).
- El mensaje se considera fuera de ventana temporal si:
(snmpEngineBoots = $2^{31}-1$) OR
(msgAuthoritativeEngineBoots $\neq$ snmpEngineBoots) OR
(msgAuthoritativeEngineTime - snmpEngineTime $> \pm 150$ seg)



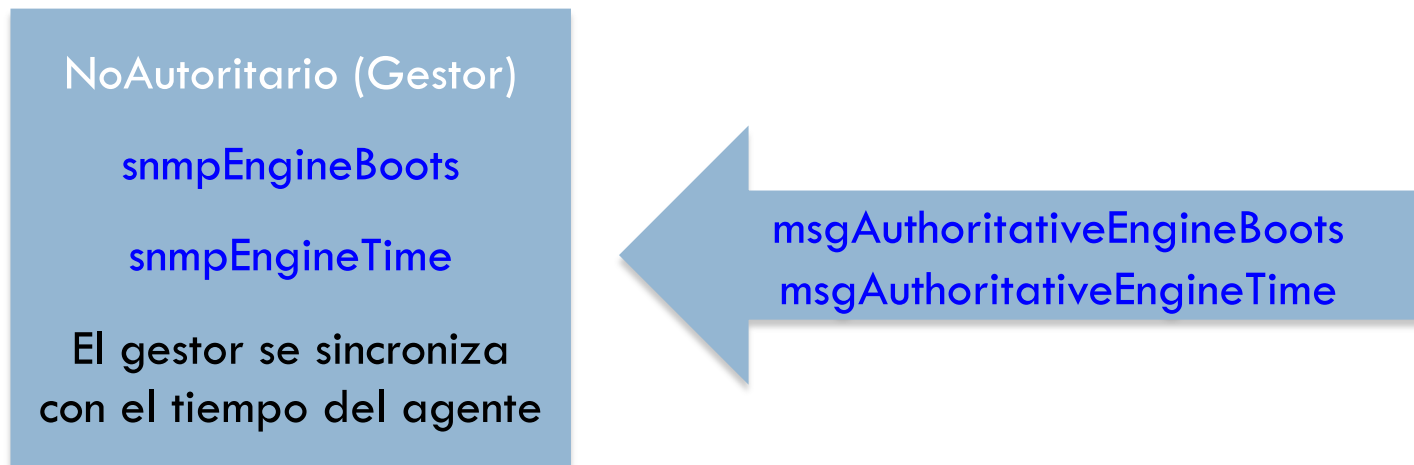
Comprobación de tiempo en receptores no autoritarios

37

- Compara los valores de snmpEngineBoots y snmpEngineTime que mantiene para un determinado engineID con los valores recibidos de dicho engineID en el mensaje (msgAuthoritativeEngineBoots y msgAuthoritativeEngineTime).

- El mensaje se considera fuera de ventana temporal si:

(snmpEngineBoots = $2^{31}-1$) OR
(msgAuthoritativeEngineBoots < snmpEngineBoots) OR
[(msgAuthoritativeEngineBoots = snmpEngineBoots) AND
(msgAuthoritativeEngineTime < snmpEngineTime - 150 seg)]



MIB usmUser

38

- Cada entidad SNMP mantiene un grupo MIB denominado usmUser:
 - ▣ usmUserSpinLock: permite coordinación para intercambio de clave
 - ▣ usmUserTable: tabla con la siguiente estructura (entrada --> usuario):
 - usmUserEngineID (I): identifica el motor SNMP.
 - usmUserName (I): nombre del usuario USM, securityName del principal.
 - usmUserSecurityName: igual pero en formato independiente.
 - usmUserCloneFrom: puntero a otra fila desde la que se clona parámetros.
 - usmUserAuthProtocol: tipo de autenticación (Noauth, HMACMD5, HMACSHA)
 - **usmUserAuthKeyChange**: permite al administrador cambiar la clave.
 - **usmUserOwnAuthKeyChange**: permite a un usuario cambiar su clave.
 - usmUserPrivProtocol: protocolo de cifrado (NoPriv, DESPriv) utilizado.
 - **usmUserPrivKeyChange**: permite al administrador cambiar la clave.
 - **usmUserOwnPrivKeyChange**: permite a un usuario cambiar su clave.
 - usmUserPublic: valor escrito en el proceso de cambio de clave, después del proceso es leído para verificar si el cambio se ha efectuado. La norma no especifica cómo se usa.
 - usmUserStorageType
 - usmUserStatus

Funciones criptográficas

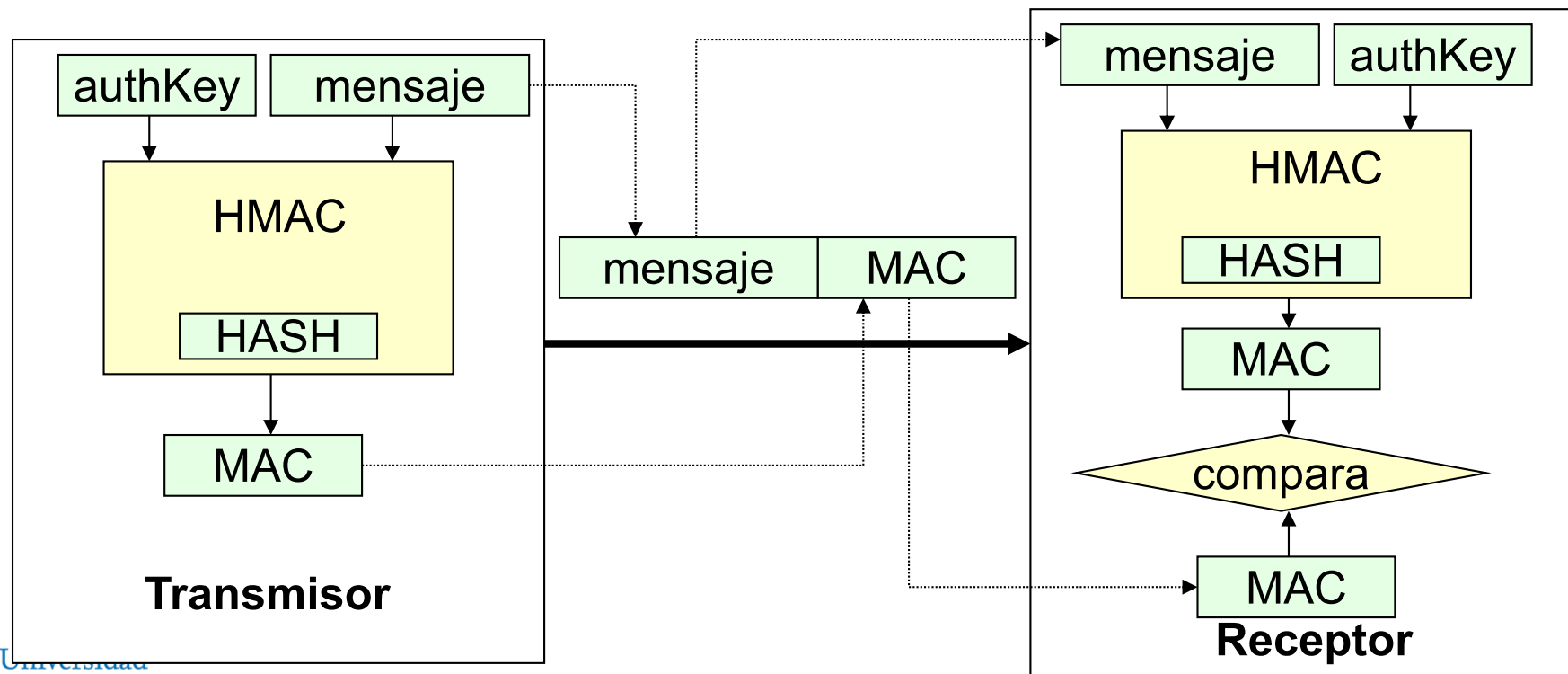
39

- Claves para privacidad privKey y autenticación authKey
- Los valores no son accesibles vía SNMP.
- Se mantiene estos dos valores para los siguientes usuarios:
 - ▣ Usuarios locales para los que son permitidas operaciones SNMP.
 - ▣ Usuarios remotos en otros motores con los que se quiere comunicar.
- Autenticación: dos posibles algoritmos
 - ▣ HMAC-MD5-96: utiliza función HMAC con hash MD5 y authKey de 128 bits. La salida de 128 bits se trunca a 96.
 - ▣ HMAC-SHA-96: utiliza función HMAC con hash SHA-1 y authKey de 160 bits. La salida de 160 bits se trunca a 96.
 - ▣ La salida de 96 bits se inserta en msgAuthenticationParameters.
- Cifrado: DES en modo CBC con clave privKey de 64 bits (también AES).
 - ▣ Generación del vector de inicialización IV:
 - pre-IV = últimos 8 octetos de los 16 octetos de privKey (secreto).
 - salt = snmpEngineBoots (4 bytes)|| 4 bytes generados localmente
 - salt se inserta en msgPrivacyParameters (público), uno distinto cada vez.
 - IV = salt XOR pre-IV (secreto)

Autenticación con MAC

40

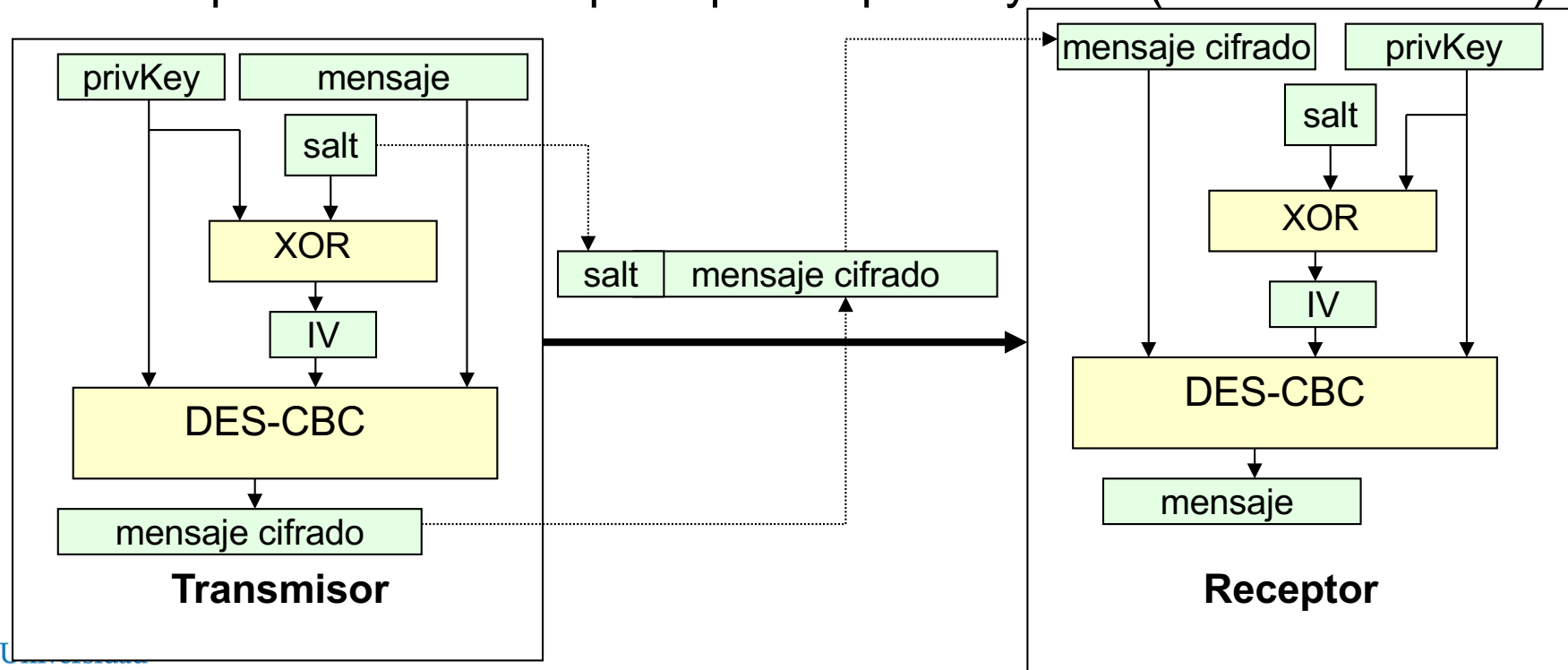
- ❑ Función HASH: saca una salida de longitud fija en función de entrada de cualquier longitud. Irreversible.
- ❑ MAC: genera un código de longitud fija en función de un mensaje de entrada y un valor secreto (authKey). HMAC utiliza funciones HASH internamente.
- ❑ Autenticación: sólo puede generar MAC quien posea authKey.



Cifrado con DES-CBC

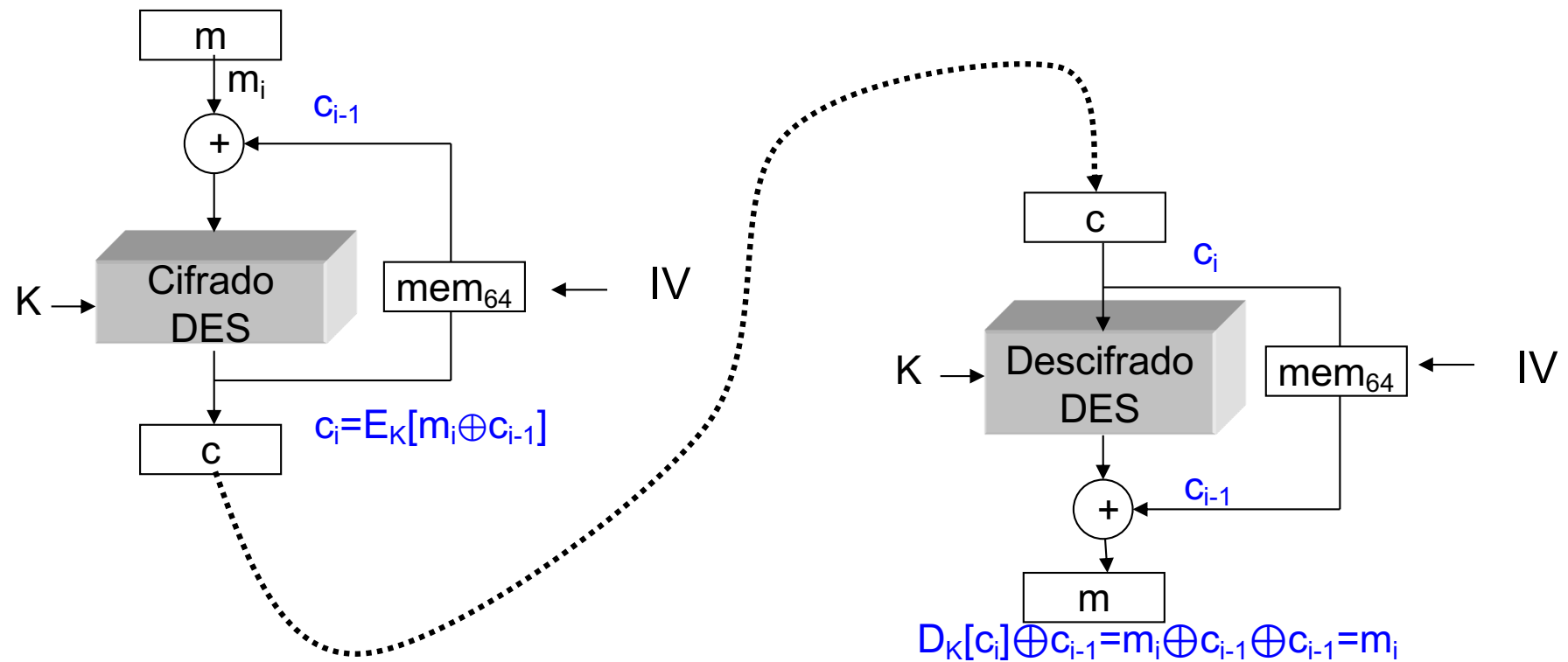
41

- Se cifra el mensaje utilizando el algoritmo DES en modo CBC.
- Para cifrar se utiliza un valor secreto (privKey) de 64 bits (56 reales).
- Para cifra con CBC necesita un valor denominado un vector de inicialización IV.
- Sólo puede descifrar quien posea privKey e IV (valores secretos).



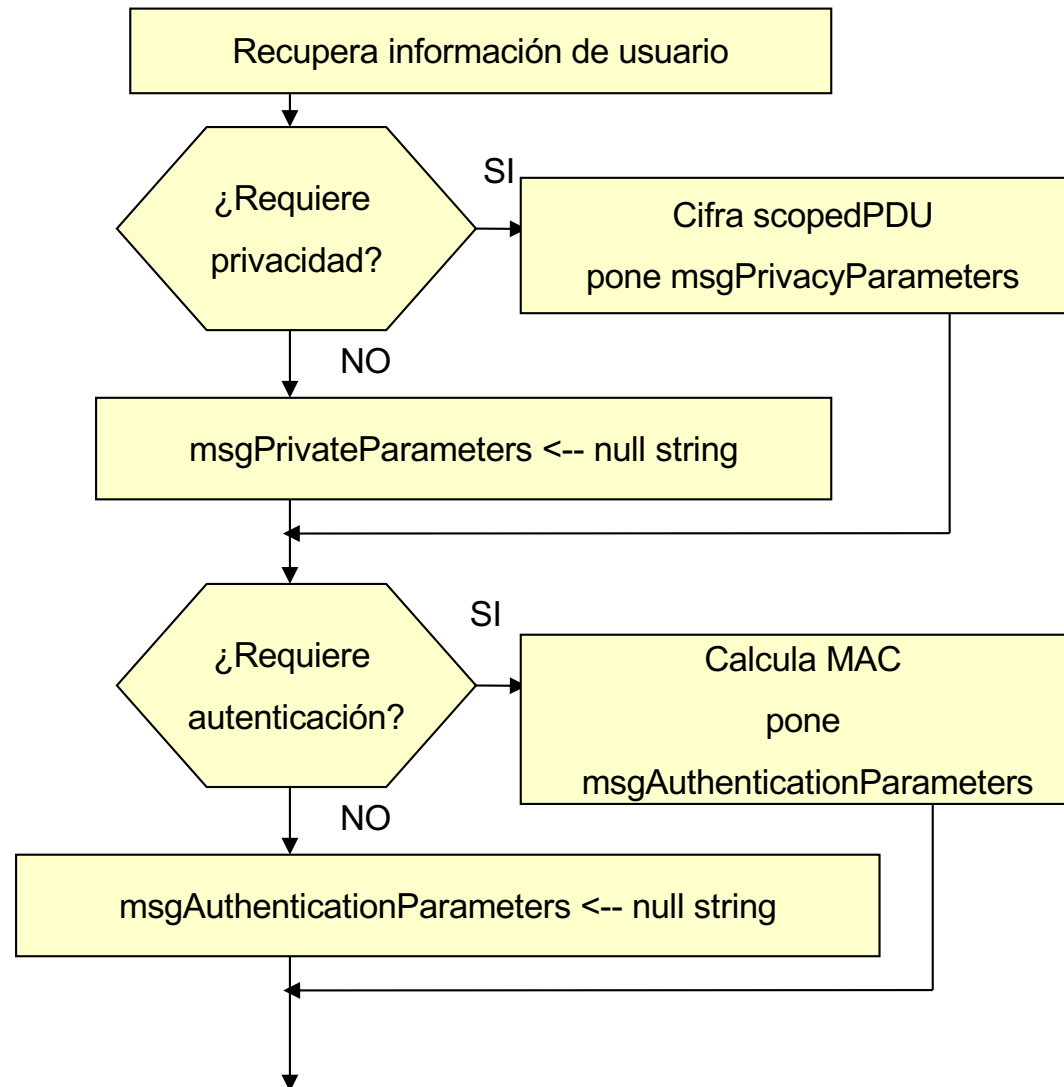
DES-CBC

42



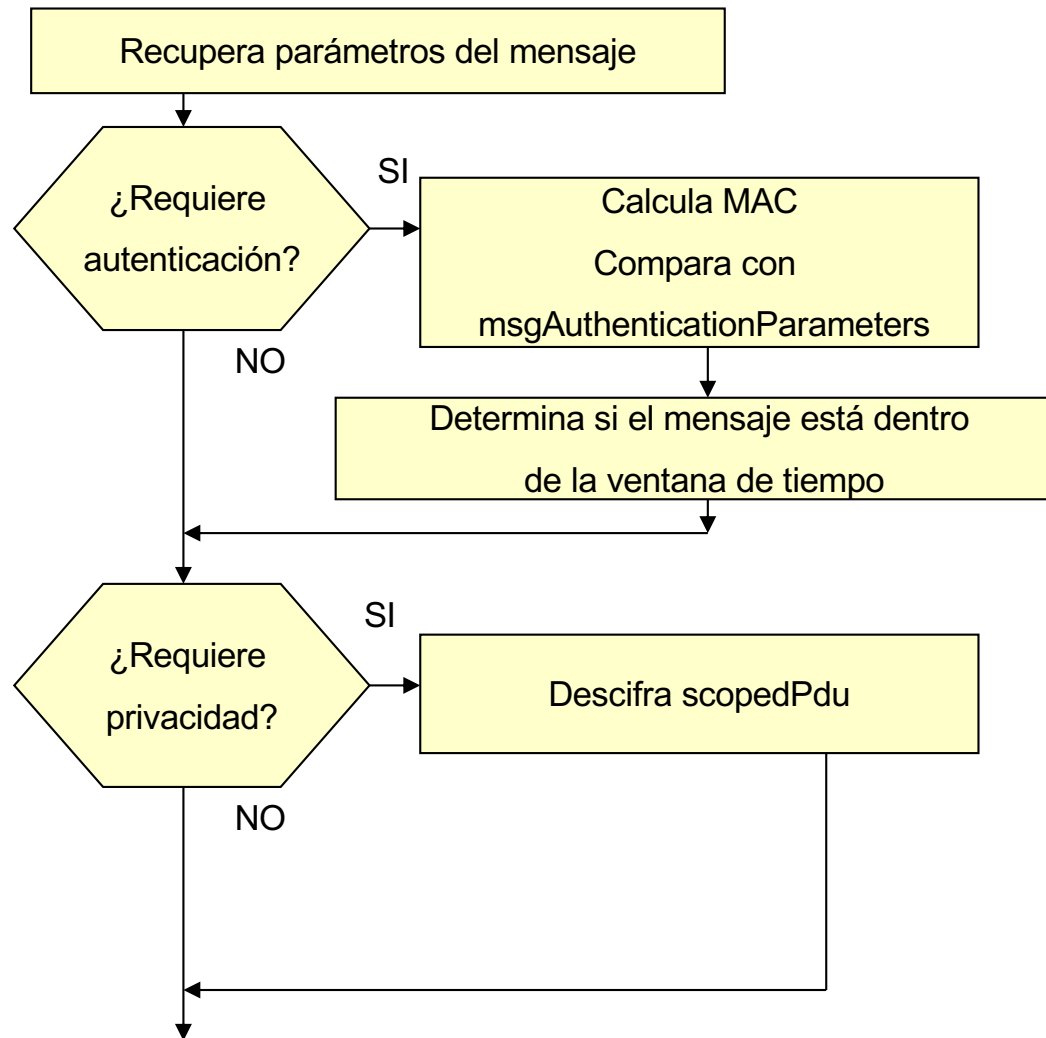
Procesamiento para enviar mensajes

43



Procesamiento al recibir mensajes

44



Procedimiento de descubrimiento

45

- Proceso utilizado por los motores no autoritario para obtener información de inicialización de los motores autoritarios:
 - ▣ Obtención de snmpEngineID:
 - Motor no autoritario -> Motor autoritario
 - Request: securityLevel = noAuthnoPriv; msgUserName=initial, msgAuthoritativeEngineID = null, varBind-List = empty
 - Motor autoritario -> Motor no autoritario
 - Report: msgAuthoritativeEngineID = snmpEngineID
 - ▣ Sincronización (autenticación necesaria):
 - Motor no autoritario -> Motor autoritario
 - Request:Autenticado (msgAuthoritativeEngineID = snmpEngineID, msgAuthoritativeEngineBoots = 0, msgAuthoritativeEngineTime = 0)
 - Motor autoritario -> Motor no autoritario
 - Report: msgAuthoritativeEngineBoots = snmpEngineBoots, msgAuthoritativeEngineTime = snmpEngineTime

Gestión de claves

46

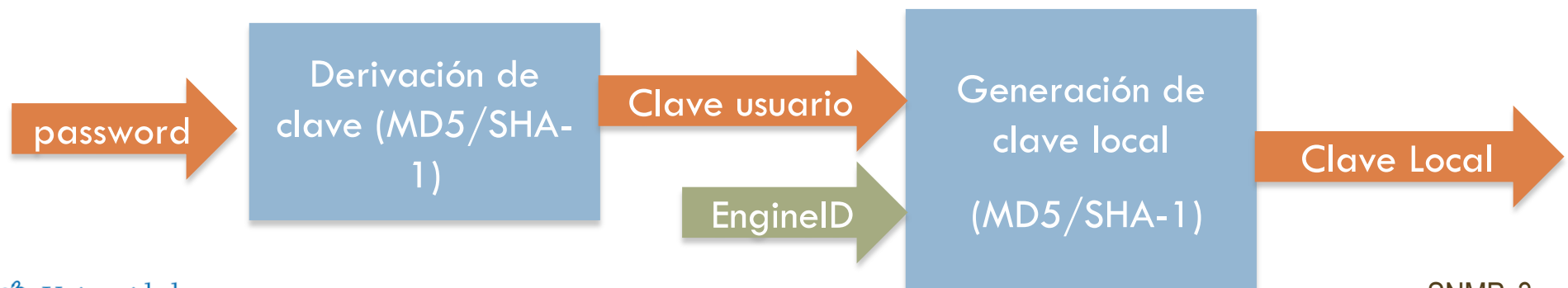
- Cada principal mantiene una clave de autenticación y una de cifrado.
- Las claves son compartidas por los motores en los que opera el principal.
- Es necesaria una **pareja de claves** (cifrado y autenticación) por cada pareja de motores (no autoritario y autoritario) para un principal dado.
- Las claves son generadas en el gestor y enviadas al agente:
 - De forma manual
 - Usando protocolo de seguridad (la norma no especifica ninguno).



Generación de claves

47

- **Clave de usuario:** derivada de la password (RFC 3414).
 - ▣ Repite la password hasta longitud de 220 octetos (añade relleno si es necesario).
 - ▣ Pasa la función hash MD5/SHA-1 para obtener claves de usuario de 128/160 bits.
- **Clave local:**
 - ▣ A partir de la clave de usuario se generan claves locales para cada agente SNMP (motor autoritario).
 - ▣ Procedimiento: se concatena EngineID del agente a la clave del usuario y se pasa la función hash (MD5/SHA-1) para obtener la clave local de 128/160 bits.



Refresco de claves

48

- SNMPv3 define el siguiente proceso de actualización de clave:
 - El solicitante genera un n° aleatorio: **random**
 - Calcula:
 - **digest** = hash(**keyOld** || **random**); hash es MD5 o SHA-1
 - **delta** = **digest** XOR **keyNew**
 - **ProtocolKeyChange** = **random** || **delta**
 - Hace un **set** sobre la tabla **usmUserTable** del agente:
 - **(Own)KeyChange** = **ProtocolKeyChange**
 - El **agente** calcula: **digest** = hash(**keyOld** || **random**)
 - **keyNew** = **digest** XOR **delta**
- ¿Quién puede actualizar claves?
 - El **administrador** de la red: clave inicial, vigila que el usuario cambie la clave periódicamente. Set en **KeyChange**.
 - El **propietario** de una clave: Set en **OwnKeyChange**.

Modelo de control de acceso VACM

Modelo de seguridad VACM

50

- Control de acceso basado en vistas definido en RFC 3415.
- Determina si el acceso a un objeto local por un principal es permitido.
- Utiliza MIBs que definen la política de control de acceso en el agente.
- Elementos del modelo VACM:
 - ▣ **Grupo:** conjunto de tuplas <**securityModel**, **securityName**> en cuyo nombre pueden ser accedidos los objetos de gestión. Identificado por **groupName**.
 - ▣ **Nivel de seguridad:** para el que se realizó la petición en el mensaje SNMP. Si se requiere **privacidad**, **autenticación**. Va en **msgFlags**.
 - ▣ **Contexto:** subconjunto de **instancias de objetos** en un MIB local. Identificado por **contextName**.
 - ▣ **Vistas de MIB:** conjunto de subárboles en una MIB.
 - ▣ **Política de acceso:** permite configurar derechos de acceso en un motor SNMP.

- El MIB VACM contiene información de las siguientes categorías:
 - **vacmContextTable** define contextos locales:
 - Lista de contextos (vacmContextName) disponibles en el sistema.
 - Utilizada por la tabla de control de acceso vacmAccessTable.
 - **vacmSecurityToGroupTable** define grupos:
 - Asigna un groupName a securityModel y securityName dados.
 - **vacmAccessTable** define derechos de acceso para grupos.
 - Contexto: contextPrefix y contextMatch (exact/prefix).
 - Grupo: nombre del grupo
 - Seguridad: securityLevel y securityModel.
 - Vista: readViewName, writeViewName y NotifyViewName.
 - **vacmMIBViews** define vistas:
 - Permite coordinar operaciones set de varios Command Generator con el objeto **spinLock**.
 - **vacmViewTreeFamilyTable**: nameView, subtree, mask, type (inc/exc).

Proceso de control de acceso

52

- Las aplicaciones Command Responder y Notification Originator consultan permisos en VACM con la primitiva `isAccessAllowed`
 - ▣ Parámetros de entrada:
 - **securityModel**: modelo de seguridad en uso SNMPv1, v2, USM.
 - **securityName**: principal que quiere acceder.
 - **securityLevel**: nivel de seguridad (privacidad, autenticación...).
 - **viewType**: tipo de acceso solicitado (lectura, escritura o notificación).
 - **contextName**: contexto que contiene `variableName`.
 - **variableName**: OID del objeto.
 - ▣ Salida: `statusInformation`; puede tomar uno de los siguientes valores:
 - **noSuchContext**: el contexto `contextName` no es soportado.
 - **noGroupName**: grupo no soportado.
 - **noAccessEntry**: no hay definidas vistas de MIB para la combinación de `securityModel`, `securityName`, `securityLevel` y `contextName`.
 - **noSuchView**: no se encuentra vista para el `viewType` especificado.
 - **notInView**: la variable no está en la MIB para el `viewType` especificado.
 - **accessAllowed**: se permite el acceso.

Proceso de control de acceso

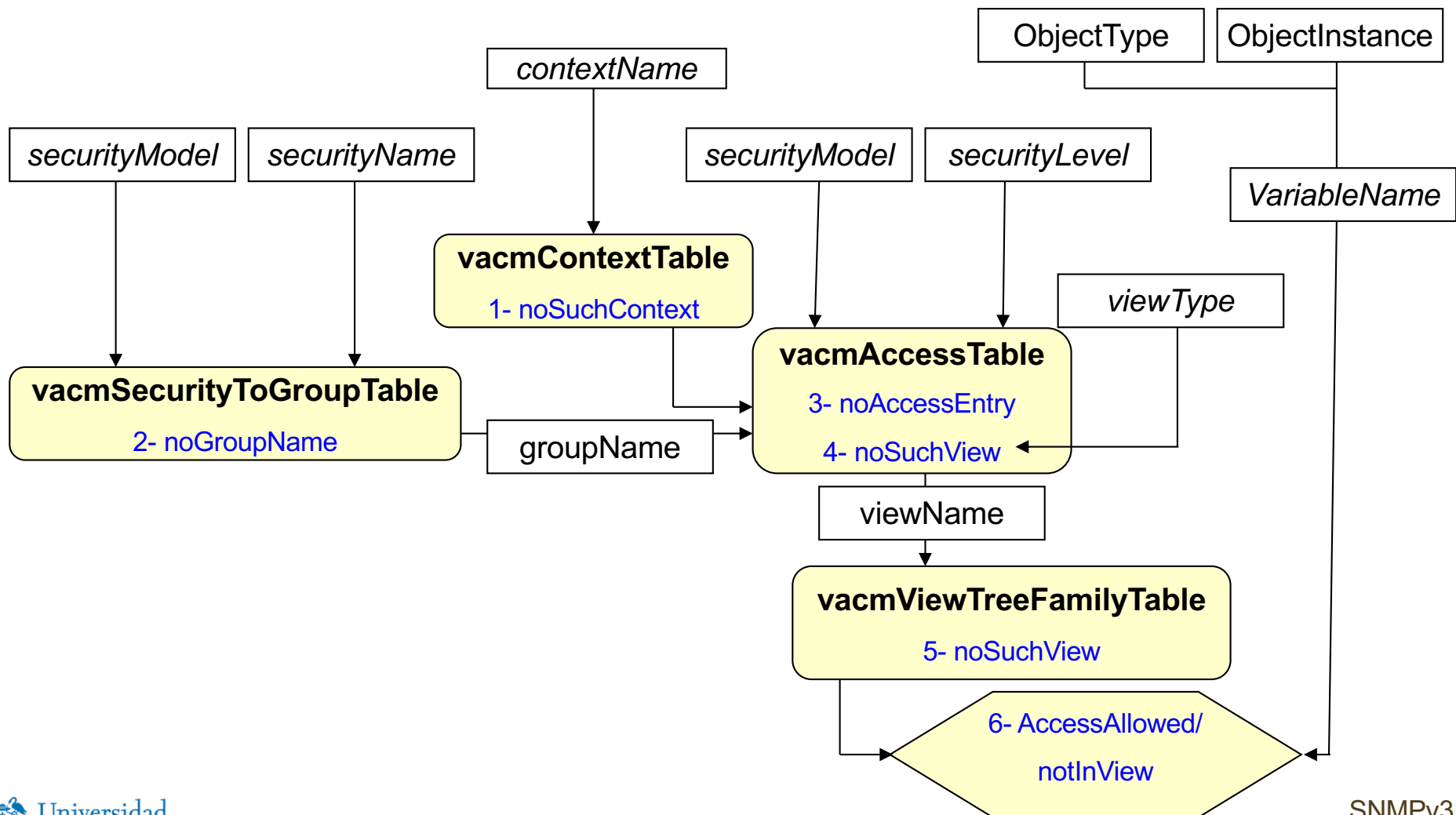
53

- Significado de los parámetros que interviene en el proceso:
 - ▣ **securityName** identifica el principal cuya comunicación está protegida por un securityModel.
 - ▣ **contextName** especifica dónde se encuentra el objeto gestionado.
 - ▣ **securityModel** y **securityLevel** especifican cómo es protegida la PDU de request o inform entrante.
 - ▣ **viewType** especifica el tipo operación para la que se solicita acceso: read, write o notify. La entrada seleccionada en vacmAccessTable contiene una viewName para cada tipo de operación, que a su vez referencia una vista en vacmViewTreeFamilyTable.
 - ▣ **VariableName** identifica el objeto gestionado (OID) y el tipo de valor que representa. Se comprueba si la VariableName pertenece a la vista seleccionada.

Proceso de control de acceso

54

- Relación entre parámetros y tablas que intervienen en la toma de decisión de acceso.



SNMPv3

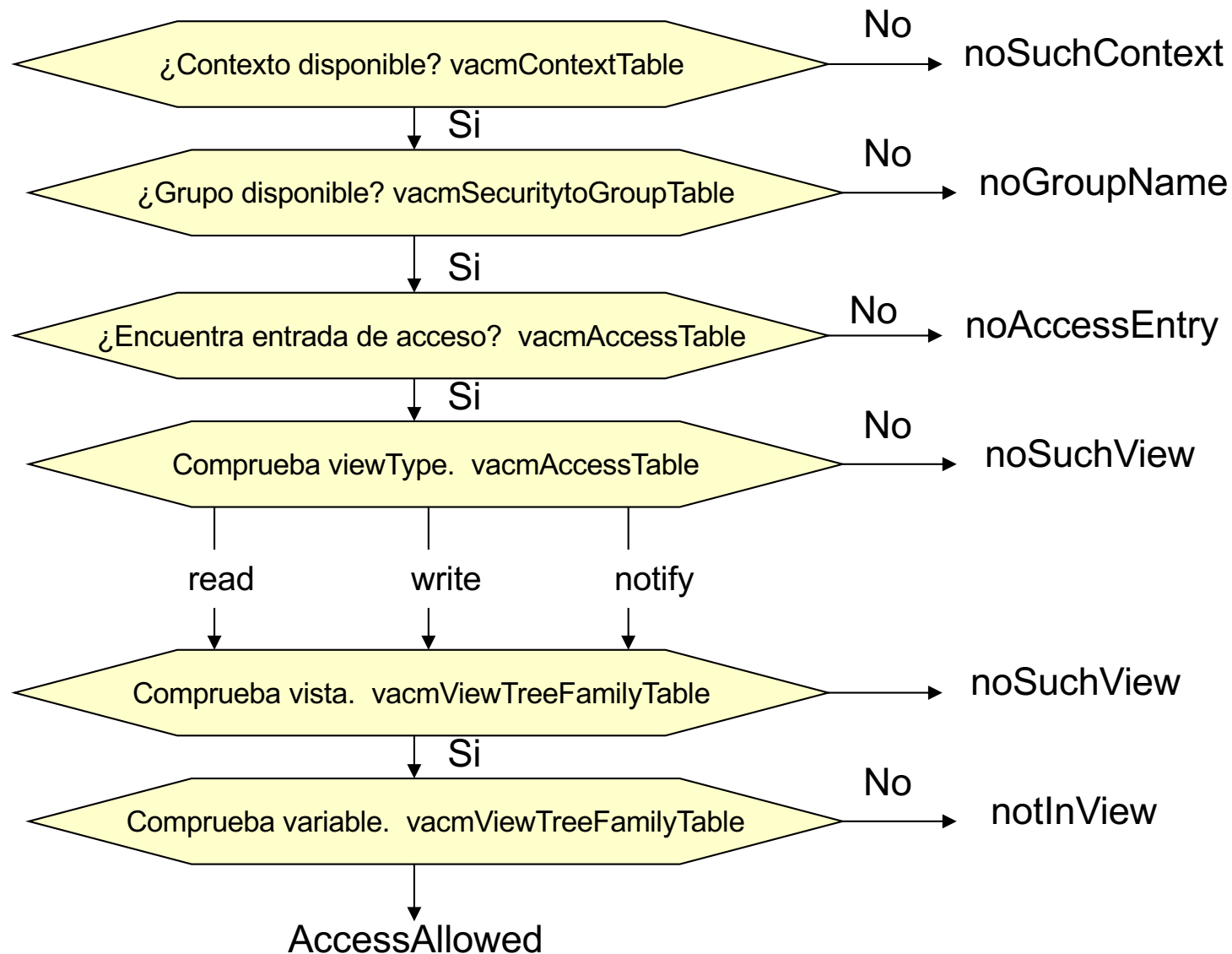
Proceso de control de acceso.

55

- Pasos a seguir para la toma de decisión de acceso.
 - ▣ VACM comprueba si hay entrada en vacmContextTable para el contextName dado. Si no devuelve el error **noSuchContext**.
 - ▣ VACM comprueba si hay un grupo en vacmSecurityToGroupTable con los valores de <securityModel, securityName>. Si no devuelve el error **noGroupName**.
 - ▣ VACM consulta vacmAccessTable con los índices groupName, contextName, securityModel y securityLevel. Si encuentra una entrada y se ha definido una política de to acceso para groupName para acceder sobre contextName bajo securityModel y securityLevel. Si no devuelve el error **noAccessEntry**.
 - ▣ VACM determina si la entrada seleccionada en vacmAccessTable hace referencia a una vista viewName con el valor de viewType. Si no devuelve **noSuchView**.
 - ▣ VACM consulta vacmViewTreeFamilyTable con el valor de viewName como índice. Si se encuentra el valor entonces ha sido configurada la vista. Si no devuelve **noSuchView**.
 - ▣ VACM comprueba que variableName pertenece a viewName seleccionada. Si no devuelve **notInView**.

Proceso de control de acceso

56



□ Referencias

- ▣ <http://www.snmp.com/snmpv3/>
- ▣ <http://www.ietf.org/html.charters/OLD/snmpv3-charter.html>
- ▣ <http://www.simple-times.org/pub/simple-times/issues/5-1.html>
- ▣ http://www.net-snmp.org/wiki/index.php/TUT:SNMPv3_Options