Capítulo 2: El modelo base

En este capítulo exponemos a detalle el trabajo de Miklau y Suciu, retomando los conceptos teóricos del modelo así como los pasos concretos que estos autores definieron para la construcción de protecciones. Con la finalidad de presentar, de la manera más clara posible, el modelo de Miklau y Suciu, ofrecemos la explicación formal de los conceptos que los autores ofrecieron en [1, 3] acompañada de anotaciones narrativas cuando resulta conveniente, siempre respetando la notación del trabajo original y empleando traducciones textuales cuando resulta necesario.

Para abordar el modelo de Miklau y Suciu presentamos, para comenzar la sección 2, la manera en la que se entienden los documentos semiestructurados como árboles de grado *n* y cuales son los conjuntos de elementos identificables en ellos. En 2.1 exponemos los diferentes tipos de llaves que Miklau y Suciu utilizaron, seguida de la noción de nodos de meta-datos en 2.2. Comenzamos la descripción de fórmulas del modelo base con la descripción del Árbol de Protección, la Función de Aceptación acc_p y las Reglas Lógicas y Protección, en 2.3, 2.4 y 2.5 respectivamente. Los conceptos concernientes al cifrado de protecciones se exponen en 2.7. La declaración de seguridad de Miklau y Suciu se presenta en 2.8. La definición y mecanismo de asignación de las Reglas de Protección se presentan, junto con las Clases de Políticas, en 2.9. Finalmente, el procedimiento que Miklau y Suciu ofrecen para crear protecciones siguiendo su modelo se expone en 2.10 y sus incisos.

2. El modelo y framework de Miklau y Suciu

En el modelo de Miklau y Suciu se trabajan los documentos XML representándolos en una estructura de árbol con nodos troncales denominados a partir del alfabeto de nombres de las etiquetas originales y con nodos hoja etiquetados a partir del alfabeto de valores originales del documento. Esto deviene del hecho de que los elementos (*tags*) de un documento semiestructurado presentan, por definición, relaciones de hermandad (*sibling*), ascendencia y descendencia que semejan, o mejor dicho, corresponden exactamente, a las que mantienen los nodos en una estructura de árbol. La única variación que existe entre las estructuras de árbol tradicionales y las de los documentos semiestructurados es que en los primeros, todos los nodos tienen una cantidad máxima uniforme de descendientes directos, mientras que en los segundos éste máximo puede estar definido por tipo de elemento o simplemente no existir. Como ejemplo simple, en un árbol binario, cada nodo puede tener, a lo más, dos nodos hijos, mientras que en un documento semiestructurado, un *tag* <sucursal> puede tener muchos descendientes <cuenta-habiente> y cada uno puede tener varios descendientes <numero-de-tarjeta>, pero hipotéticamente, se podría restringir a un máximo de 5 tarjetas por cuenta-habiente.

Para un árbol *t* construido a partir de un documento *d*, se puede definir:

- VALUES (*t*) .- conjunto de valores originales de *d* (alfabeto de etiquetas de *t* para nodos hoja).
- NODES (*t*) .- conjunto de nodos, troncales y hojas, del árbol.
- EDGES (*t*) .- conjunto de aristas entre los nodos de *t.*
- VALUE (*i*) .- etiqueta del nodo i (valor del *tag* original de *d*).

Para denotar la jerarquía de los nodos se emplea la siguiente notación:

- $i \prec j$ para i, j ϵ NODES (*t*), cuando i es ancestro estricto de j.
- $i \preceq j$ para i, j ϵ NODES (*t*), cuando i es ancestro o el propio j.

La protección sobre un árbol *t*, es otro árbol en el que los nodos se encuentran protegidos por llaves o conjuntos de llaves que controlan a nivel conceptual el acceso a cada nodo. Dado que la protección real se implementa mediante procesos criptográficos, que usualmente no permiten el uso de expresiones compuestas de varias llaves, Miklau y Suciu proponen el uso de llaves internas y meta-nodos para generar, como se explicará en la sección 2.5, una protección normalizada equivalente para *t*. En una protección normalizada, cada nodo es protegido solo por una llave, de modo que se puede someter a procesos de cifrado real para general el documento listo para publicación.

La figura 2, muestra una protección P y una protección normalizada P' para un mismo documento e integra los conceptos que se trabajarán, poco a poco, en el resto del capítulo; se puede ver el árbol original con las llaves que el autor definió para cada elemento, se puede apreciar la existencia, uso y normalización de los operadores lógicos entre llaves, etc.

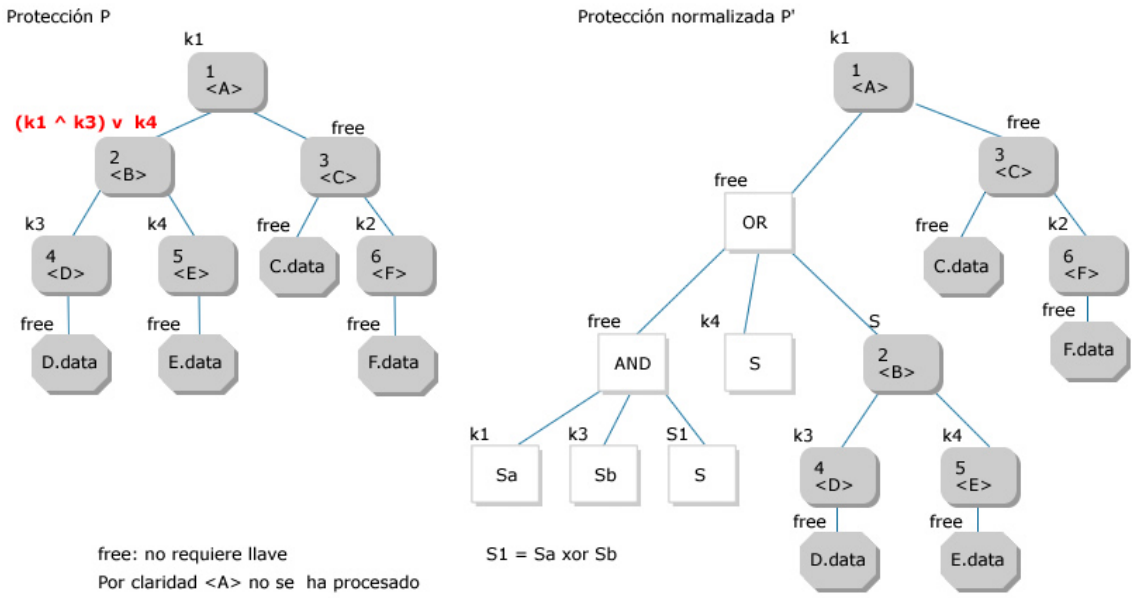

Figura 1: Protección P y protección normalizada P'
(basado en lo expuesto por Miklau y Suciu [1])

2.1 Clasificación de llaves

Miklau y Suciu proponen el uso de tres clases de llaves para los procesos de cifrado:

- Llaves de intercambio.- definidas, almacenadas y distribuidas por el productor del documento. Se les asigna un nombre público que las identifica.

  XCHGKEY es el dominio finito de llaves de intercambio

- Llaves de valor de datos.- obtenidas a partir de un campo de datos o del XOR de varios campos de datos del documento original. Se justifica su declaración explícita para los casos en los que se desee emplear los datos contenidos en el documento para definir accesos sobre el mismo.

  DATAVALUE es el dominio de valores de dato

- Llaves internas.- conceptualmente similares a las llaves *derivadas* planteadas por Gifford [2], son empleadas para hacer posible la protección con los operadores OR y AND. En el trabajo de Miklau y Suciu éstas son secuencias aleatorias generadas automáticamente por el sistema durante la creación del árbol de protección y se almacenan directamente en el documento cifrado, es decir, solamente se pueden obtener utilizando llaves de intercambio o de valor de datos para explorar el árbol hasta acceder a los meta-nodos hoja que las contengan.

INNERKEY es el dominio finito de llaves internas

De acuerdo con lo anterior, Miklau y Suciu definen el conjunto finito de llaves:

$$\text{KEY} = \text{XCHGKEY} \cup \text{INNERKEY} \cup \text{DATAVALUE}$$

2.2 Nodos de meta-datos

Cuando se añaden al árbol original de documento t nodos con valores pertenecientes a INNERKEY u operadores booleanos para construir el árbol de protección (llamados en conjunto, meta-nodos), se obtiene un meta-árbol t_m, en el que los valores de los nodos hoja pueden ser valores del documento original o llaves internas, es decir:

$$\text{VALUE}(i) \in \text{INNERKEY} \cup \text{DATAVALUE},\ \forall\ i \in \text{NODES}(t_m) \mid i \text{ es hoja}$$

En general, los nodos de t_m son un subconjunto de la unión de los nodos de t, las llaves internas y operadores lógicos utilizados para normalizar la protección. Más formalmente:

$$\text{NODES}(t_m) = \text{NODES}(t) \cup \text{METADATA}$$

para

$$\text{METADATA} = \text{INNERKEY} \cup \text{LOPNODES}$$

Donde

LOPNODES son nodos que representan a los operadores lógicos OR y AND y que se utilizan para relacionar los elementos especificados por las reglas lógicas de protección.

2.3 Árbol de protección

La protección de un árbol t se define como P = (t_m, σ) donde t_m es el árbol de metadatos y σ es el conjunto de reglas booleanas σ_i asociada a cada nodo i sobre KEY.

Las reglas de σ siguen la gramática:

$$\sigma \;:\; \mathit{true} \mid \mathit{false} \mid k \mid \sigma \wedge \sigma \mid \sigma \vee \sigma$$

donde

k: es una llave tal que $k \in$ KEY,

true: otorga acceso sin requerir una llave y

false: restringe el acceso sin excepción.

Resulta evidente que para acceder a un nodo *i*, se debe contar con un conjunto de llaves suficiente para satisfacer σ_i, sin embargo, cabe observar que para alcanzar cualquier nodo se debe pasar por todos los nodos ancestros de éste en el camino directo desde la raíz.

Considerando lo anterior, Miklau y Suciu definen la fórmula de necesidad para un nodo *i* de la siguiente manera:

$$\varphi_i = \bigwedge_{j \preceq i} \sigma_j$$

es decir, la conjunción de todas las reglas σ actuantes sobre el nodo *i* y todos sus ancestros.

La fórmula de necesidad es la regla lógica que asocia todas las llaves involucradas en el camino para alcanzar un nodo, por la manera en la que está construida y por la explicación que de ella ofrecen Miklau y Suciu en [1], se podría pensar que un usuario con un conjunto de llaves *K* que no satisfaga φ_i, no podrá acceder al nodo en cuestión ya que para poder acceder a un nodo *i* primero se debe satisfacer todas las reglas de protección de los ancestros de *i*, sin embargo, como se verá en la sección 2.10.3 esto no es necesariamente así en la implementación que los autores construyeron para su modelo.

2.4 La función de aceptación acc_p

Miklau y Suciu definen la función iterativa $acc_p\ (K)$ para obtener aquellos nodos de *t* que un usuario conociendo un conjunto de llaves *K* tenga derecho a obtener.

Primero definen:

K: conjunto de llaves conocidas inicialmente por el usuario.

$$K \subseteq \text{XCHGKEY} \cup \text{DATAVALUE}$$

M: conjunto "actual" de llaves conocidas (al inicio de acc_p, $M = K$, pero idealmente crece con el tiempo)

$$M \subseteq \text{XCHGKEY} \cup \text{DATAVALUE} \cup \text{INNERKEY}$$

N: conjunto "actual" de nodos accesibles al usuario (inicialmente $N = \{\}$)

En cada iteración de acc_p se incrementa M con los valores de las hojas disponibles en N y se incrementa N con los nodos que se pueden liberar usando M. Cuando se presenta una iteración en la que N y M se mantienen constantes se sabe que N contiene ya todos los nodos que el usuario tiene derecho a conocer.

Para liberar un nodo de su protección se pueden usar directamente las llaves en M o combinarlas para generar nuevas. La definición exacta de las reglas de combinación de llaves puede variar, de manera que Miklau y Suciu proponen que éstas sean plasmadas en una función *combine (M)* dependiendo de los protocolos y comportamiento deseados, pero siempre teniendo cuidado de que la combinación se mantenga dentro de límites computacionales realistas. La función de combinación, cualquiera que sea su implementación debe construirse de manera que para $M \subseteq \text{KEY}$ y $M' = combine(M)$, M' sea el conjunto de todas las combinaciones permitidas de los elementos de M, de modo que $M \subseteq M'$.

Miklau y Suciu adoptaron la siguiente definición:

$$combine(M) = M \cup \{ r \oplus r' \mid r, r' \in M \cap \text{INNERKEY} \} = M'$$

en la que escogieron la operación de XOR (denotada por $\oplus$) para crear una nueva llave a partir de dos llaves internas previamente conocidas.

2.5 Reglas lógicas y protección

Para denotar que un conjunto de llaves satisfacen una cierta regla se emplea la notación

$$M \models \varphi$$

Donde φ es una fórmula lógica positiva perteneciente a una protección y que obtiene un valor de verdadero si se toman como verdaderas las variables proposicionales correspondientes a las llaves contenidas en *M*, mientras se dejan como falso todas las demás. Por ejemplo:

$\{k_1, k_2, k_3\} \models k_4 \vee (k_1 \wedge k_2)$ es cierto

$\{k_1, k_2\} \models k_1 \wedge k_3$ es falso

Usando lo anterior, Miklau y Suciu definen formalmente lo expuesto para la función acc_p *(K):*

$$acc_p (K) = N \cap \text{NODES} (t)$$

Donde

$N \subseteq \text{NODES} (t_m)$ y $M \subseteq \text{KEY}$

Son el máximo conjunto obtenido de las funciones mutuamente recursivas:

$N_{h+1} = \{i \mid i \in \text{NODES} (t_m), \text{combine} (M_h) \models \varphi_i\}$ y

$M_h = K \cup \{\text{value} (i) \mid i \in N_h\}$

Para las que $N_0=\{\}$ y $M_0=K$

Miklau y Suciu declaran formalmente la equivalencia entre dos *protecciones* P y P' como:

$$(P \equiv P') \text{ sii } \forall K \subseteq XCHGKEY \cup DATAVALUE, acc_p(K) = acc_{p'}(K)$$

Sin embargo, Miklau y Suciu aclaran que el que las *protecciones* P y P' sean equivalentes no significa que sean idénticas, pueden contener diferentes meta-nodos, que es el caso de las *protecciones* normalizadas derivadas de una protección original, caso en el que cabe la figura 2.

2.6 Normalización de una protección

Antes de cifrar una protección sobre un árbol *t* se debe normalizar, es decir, las reglas de restricción que contengan dos o más llaves relacionadas con operadores lógicos $\wedge$ y $\vee$ se deben transformar en reglas atómicas relacionadas por meta-nodos apoyándose en llaves internas cuando sea necesario.

Reescritura de reglas propuesta por Miklau y Suciu para la normalización de una protección:

$k_1 \vee k_2$ sobre un nodo se normaliza creando una llave interna k' y almacenándola 2 veces, una vez bajo k_1 y otra bajo k_2 como se muestra en la figura 3.

Tras la normalización, si se desea acceder al nodo A, se debe poseer la llave k_1 o la llave k_2, con las que se podrá obtener k', que a su vez, dará acceso a A.

$k_1 \wedge k_2$ se normaliza creando dos llaves internas k_a y k_b, que se almacenan cada una bajo una de las llaves originales. Finalmente, como se muestra en la figura 4, el nodo original se protege bajo una tercera llave interna, calculada como el XOR de k_a y k_b.

Una vez reescrita la protección, si se intenta alcanzar A, se deberá contar tanto con k_1 como con k_2, obtener k_a y k_b, aplicar la operación de XOR a estas últimas para obtener k', y más tarde emplear k' para liberar a A de su protección.

Al realizar la normalización mediante reescrituras de este tipo se mantiene la intención original de cada regla y al mismo tiempo se prepara el nuevo árbol t_m para poder ser sometido a un proceso criptográfico estándar, que use una llave para cada elemento que se deba cifrar.

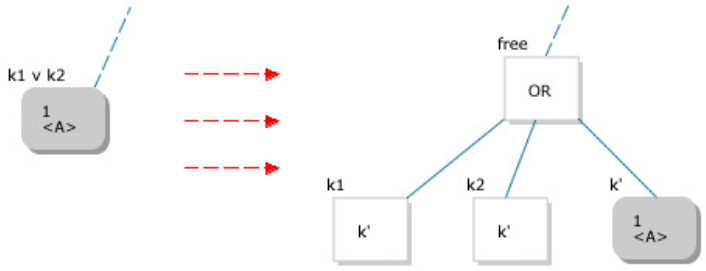

Figura 2: Normalización de una regla disyuntiva

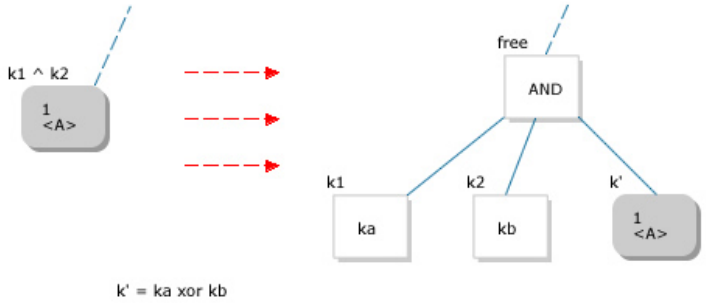

Figura 3: Normalización de una regla conjuntiva

2.7 Cifrado de la protección

Tras normalizar una protección P, la implementación del cifrado se realiza como un proceso *bottom-up* desde los nodos hoja hasta la raíz en el que los segmentos previamente cifrados son contenidos en otros más generales. Para lograr esto, el *framework* de Miklau y Suciu se apoya de la recomendación del W3C [7] para representar los elementos cifrados en el cuerpo del documento XML.

2.7.1 El algoritmo

La construcción del árbol cifrado a partir de una protección normalizada P', en la que cada elemento está protegido por una regla atómica, se lleva a cabo con "un recorrido recursivo transversal" [1] de ésta, es decir, se explora el árbol desde las hojas hacia la raíz en un proceso recursivo común modificando cada nodo del árbol de acuerdo a su protección específica:

FREE: Se mantiene en su estado actual (Miklau y Suciu utilizan TRUE para denotar esta protección [1]).

FALSE: Se elimina el elemento completo, incluyendo sus descendientes.

k_x: Se traduce en un elemento cifrado tras someter el nodo completo, con sus descendientes, a un proceso criptográfico usando la llave k_x.

2.7.2 Representación de elementos

Miklau y Suciu indican que cada elemento cifrado del documento se represente, siguiendo los lineamientos del W3C, mediante un elemento *EncryptedData*, que contenga los siguientes sub-elementos:

- *EncryptionMethod*: en el que se coloca información acerca del algoritmo y parámetros empleados en el cifrado.

  En su trabajo experimental, Miklau y Suciu se limitaron a utilizar AES de 128 bits (El AES o Advanced Encryption Standard es, actualmente el algoritmo Rijndael y es capaz de operar con llaves de una longitud de 128 o 256 bits).

- *KeyInfo*: que contiene la descripción y otros datos acerca de la llave empleada para el cifrado / descifrado. Contiene los siguientes elementos:
  - *Name*: en el caso de referirse a una llave de intercambio tiene una sola ocurrencia, indicando el nombre de la llave. Cuando se refiere a llaves internas puede ocurrir una sola vez, con el nombre de

la llave interna, o varias, con los nombres de las llaves internas que, mediante una operación de XOR, producirán la llave necesaria para descifrar el elemento actual.

- *Path*: para los casos de llave de valor de datos, indica el patrón de XPath que indica el origen de la llave.
- *Concealment*: para los casos de llave de valor de dato, contiene el resultado de $v \oplus r$, donde r es una cadena aleatoria y v es el valor de la llave de dato.
- *Hash*: para las llaves de valor de dato, especifica h(r), el resultado de aplicar una función *hash* a la cadena aleatoria r usada en el elemento anterior[1].

- *CipherData*: contiene la salida del proceso de cifrado, representada en texto a base de caracteres codificados en 64 bits.

- *EncryptionProperties*: reservada para información adicional opcionalmente definida por el usuario.

2.8 Declaración de seguridad

Miklau y Suciu argumentan que el modelo propuesto genera protecciones que mantienen la siguiente propiedad (traducida textualmente de [1]):

[1] Miklau y Suciu afirman en [1] que es computacionalmente difícil derivar v a partir de los valores proporcionados en *concealment* y *hash*, sin embargo es eficiente probar si un valor propuesto v' es igual a v usando esta información.

Supóngase que t es un documento XML, P es una protección sobre t, y t' es la implementación de P sobre t descrita arriba. Entonces, para cualquier conjunto de llaves K:

1. Si $x \in \mathrm{acc}_P(K)$, entonces existe un algoritmo eficiente para reproducir x a partir de t'.
2. Si $x \notin \mathrm{acc}_P(K)$, entonces aún el algoritmo más eficiente para reproducir x requiere "adivinar" las llaves faltantes.

La demostración del primer punto de la propiedad se desprende directamente de la definición de la función $\mathrm{acc}_P(K)$, de la que fácilmente se pueden construir implementaciones eficientes. El segundo punto, sin embargo, quedó sin demostración en el trabajo de Miklau y Suciu, en el que simplemente declararon que éste dependía de la suposición de que cada protocolo de cifrado empleado fuera, en sí mismo, seguro. Además, argumentaron que la demostración de que la combinación de protocolos se mantenía segura quedaba fuera del alcance de dicho trabajo; sin embargo, en el año 2005, Abadi y Warinschi analizaron el modelo de Miklau y Suciu desde "el contexto de los rigurosos modelos de la criptografía moderna" y lograron demostrar en [4] "que si cierta información se puede considerar secreta de acuerdo con la semántica abstracta de las protecciones, entonces ésta está, también, computacionalmente escondida", por lo que el segundo punto de la propiedad de seguridad de Miklau y Suciu quedó demostrado. Cabe resaltar, sin duda, que Abadi y Warinschi decidieron excluir de su análisis el uso de las llaves de valor de datos, por considerar que la entropía de éstas podría no ser suficiente

para evitar que fueran adivinadas en determinados casos; se puede decir que la decisión de incluir o no este tipo de llaves depende en gran medida del conocimiento del autor sobre la distribución de los datos y del contexto en el que se quiera proteger la información, ya que será esto lo que determine la probabilidad de que las llaves de este tipo puedan ser adivinadas fácilmente.

Se puede decir, de acuerdo a las demostraciones de Miklau y Suciu [1] y de Abadi y Warinschi [4], que el modelo de protecciones es seguro en cuanto a que la información protegida no puede ser revelada, en ausencia de un conjunto satisfactorio de llaves, sin la dificultad computacional que implica adivinar las llaves faltantes. Sin embargo, es cierto que existen deficiencias en el modelo en cuanto a que un adversario puede obtener datos acerca de la información protegida aún sin romper la protección. Específicamente se reconoce en [1] que un adversario puede conocer el número de hijos protegidos presentes en un elemento determinado, ya que cada descendiente protegido se cifra por separado y se expone como un elemento *EncryptedData* independiente, por lo que también es posible aprender el tamaño de los subárboles cifrados; además, de acuerdo a Miklau y Suciu [1], se pueden intuir aspectos de la política de seguridad a partir de los nodos de meta-datos presentes en el árbol.

2.9 Reglas de protección

La definición de una protección *P* esbozada hasta el momento habla, en términos generales, de un conjunto de reglas σ escritas sobre el conjunto de llaves KEY y asignadas a los elementos del árbol *t*; se puede intuir, entonces, que es necesario contar

con un mecanismo mediante el cual el autor del documento pueda determinar las reglas que habrán de actuar sobre cada elemento de *t*, estas asignaciones dependerán de las políticas de acceso que el autor desee implementar sobre el documento.

Se puede intuir, también, que cada política está dirigida a un elemento objetivo específico, sin embargo, dado que es posible que se deba observar la misma política de acceso para varios elementos con características similares en un documento, Miklau y Suciu decidieron permitir que cada política de acceso tuviera uno o varios objetivos sobre los que la política intentará imponer controles. Miklau y Suciu eligieron utilizar el lenguaje de consulta XQuery como mecanismo para dictar las políticas de acceso, la manera en la que lo hicieron se explica más adelante, en la sección 2.10.1.

Para materializar el control de acceso, cada política necesita definir un conjunto de condiciones que se deberán satisfacer antes de poder acceder al o a los elementos protegidos. Dado que el modelo de Miklau y Suciu se basa en el cifrado, todos los controles, es decir, todas las políticas, estarán definidas a partir de la posesión o derivación de llaves criptográficas y, dada la flexibilidad del modelo, una sola política podrá condicionar a la posesión de más de una llave. En su trabajo, Miklau y Suciu decidieron permitir solo relaciones conjuntivas al interior de las políticas, es decir, si una política se define en función de más de una llave, la satisfacción de la política se dará solo con la posesión de todas y cada una de las llaves.

2.9.1 Clases de políticas

Miklau y Suciu consideraron conveniente permitir dos clases de políticas que pudieran ser aplicadas en la construcción de una regla de protección:

- Las políticas de suficiencia.- son todas aquellas cuya satisfacción es credencial bastante para obtener el acceso a lo que se intenta proteger.

- Las políticas de necesidad.- cuya satisfacción es obligatoria para conseguir el acceso al elemento resguardado.

Se puede decir que si el autor del documento decidiera definir el acceso a un elemento *e* por un conjunto de políticas y todas fueran políticas de suficiencia, cualquier usuario que pudiera satisfacer al menos una de esas políticas tendría derecho a acceder al elemento *e*. Por el contrario, si el autor condicionara el acceso a *e* con un conjunto conformado únicamente por políticas de necesidad, el usuario debería de satisfacer todas las políticas antes de acceder al elemento deseado. Dado el comportamiento que Miklau y Suciu plasmaron en las reglas de necesidad y de suficiencia, es fácil observar que mezclar las dos clases de políticas puede generar complicaciones. A consecuencia de lo anterior, se dice que la protección *P* que soporte todas las políticas de acceso al mismo tiempo puede no existir; en otras palabras, dado un conjunto de políticas de acceso definidas por el autor para un documento *d*, existirá una protección *P* que respalde todas las políticas si y sólo si, éstas son consistentes entre sí.

2.10 Procedimiento de construcción de una protección

"Revisar la consistencia de las políticas y construir la protección necesaria no es trivial" [1], por lo que Miklau y Suciu definen un procedimiento que puede separarse en cuatro pasos totales: definir las políticas de acceso, traducirlas a reglas primitivas, construir la protección y, finalmente, cifrar el documento.

2.10.1 Definición de políticas de acceso

Aprovechando el trabajo existente en el tratamiento de documentos XML, Miklau y Suciu decidieron hacer una adaptación del lenguaje de consulta XQuery para definir las políticas de acceso en su modelo de protecciones. Las políticas para los elementos específicos se expresan a través de *policy queries* que el autor del documento definirá y aplicará sobre el documento en claro en un ambiente seguro. Los *policy queries* siguen la siguiente estructura:

[SUFFICIENT | NECESSARY]

FOR… LET… WHERE…

KEY [$keyExpr_1$, …]

TARGET [$targetExpr_1$, …]

Donde:

- SUFFICIENT | NECESSARY determina a cuál de las clases pertenece la política.
- FOR… LET… WHERE… son cláusulas que actúan exactamente como en XQuery
- $keyExpr_i$ se entiende como :

  [path-expr] | [getKey(*keyName*) [keyChain (*keyChainName*)]]

Donde:

- path-expr se refiere a la ubicación de un elemento que habrá de ser usado como llave de valor de datos.
- getKey(*keyName*) se refiere al uso de una llave de intercambio, identificada con el nombre definido en *keyName*, que de existir se recuperará del repositorio para ser empleada y de no hacerlo se generará automáticamente para luego almacenarse. Opcionalmente se puede ligar una llave a un grupo de llaves asociadas, identificado por *keyChain*.

Pueden incluirse tantos *keyExpr* como se deseen en cada política, pero se debe recordar siempre que todas las llaves incluidas en éstas deberán poseerse para satisfacer la política específica al momento de consultar el documento protegido.

- $targetExpr_i$ es una expresión en XPath que identifica un elemento al que se aplicará la política, puede haber tantos elementos *targetExpr* como se deseen, pero al menos cada política debe contar con uno.

(A partir de este momento se hará un uso indistinto de "política" y "*policy query*".)

2.10.2 Construcción de reglas primitivas

Una vez definidos los *policy queries*, el siguiente paso propuesto por Miklau y Suciu es calcular todos los nodos a los que cada política se aplica, es decir, todos aquellos que se puedan seleccionar a partir de las expresiones en la sección de TARGET.

De acuerdo con Miklau y Suciu, el cálculo de los elementos implicados, tanto en TARGET como en KEY, por las cláusulas FOR, WHERE y LET de las expresiones de XQuery empleadas en las *policy queries* es "un paso estándar en cualquier procesador de XQuery", sin embargo, el tratamiento de ese lenguaje está fuera del alcance de este documento, por lo que se omite la descripción del proceso.

Por cada *policy query*, a cada elemento e seleccionado en TARGET y a todo elemento $e' \mid e \prec e'$ se asocian las expresiones o reglas primitivas:

$r_s = \{k_1, k_2, k_3, \ldots\} \rightarrow e$ si se trata de una política SUFFICIENT

y

$r_n = e \rightarrow \{k_1, k_2, k_3, \ldots\}$ si se trata de una política NECESSARY

Cuyo uso e interpretación se verá más adelante y en las que $\{k_1, k_2, k_3, \ldots\}$ son las llaves indicadas en la sección KEY de la política y se pueden representar usando el conjunto S, es decir, $S = \{k_1, k_2, k_3, \ldots\}$ para cierta política r.

Se añaden las reglas primitivas de protección a todos los descendientes de los elementos e siguiendo la intuición de que para acceder a un nodo $e' \mid e \prec e'$ donde e se encuentra protegido.

2.10.3 Construcción de la protección

Dada la semántica de las políticas de suficiencia y de necesidad, Miklau y Suciu argumentan que no existe una única protección P_R que satisfaga el conjunto de reglas R sobre t y ofrecen el siguiente ejemplo:

> Teniendo un árbol *t* y un conjunto de reglas de protección *R*, si *R* fue derivada solamente de políticas de suficiencia, una protección libre en la que ningún elemento requiera una llave para ser alcanzado, satisface a *R*.

Dicho de otra forma, la semántica de las políticas de suficiencia dicta que cualquiera que pueda satisfacer una de las regla derivadas para un elemento determinado, deberá tener acceso él; en una protección que no requiere llaves, esta condición se cumple siempre, es decir, todos aquellos que, con su conjunto de llaves, puedan satisfacer una de las reglas derivadas de una política de protección sobre cierto elemento, podrán tener acceso a él ya que ningún elemento, protegido o no por una política de suficiencia, requiere llave alguna para revelarse, por esto se dice que la protección libre satisface a *R*.

El ejemplo puede parecer carente de sentido práctico, ya que es completamente fútil proteger un documento de manera que aquellos individuos que posean los conjuntos de llaves adecuados puedan accederlo, si al mismo tiempo cualquier otro individuo con un conjunto de llaves incompleto o nulo puede accederlo también; sin embargo el planteamiento de Miklau y Suciu ayuda a demostrar que el objetivo no debe ser solamente satisfacer todas las reglas en *R*, sino construir la protección más restrictiva posible que aún satisfaga *R*.

Formalmente se define lo siguiente:

Se dice que una protección *P* satisface a un conjunto de reglas *R*, (denotado por $P \triangleright R$), si:

$\forall\ r_s \in R \mid r_s = S \rightarrow e$, entonces $e \in acc_p(S)$

y

$\forall\ r_n \in R \mid r_n = e \rightarrow S$ y $\forall\ K \mid e \in acc_p(K)$, entonces $S \subseteq K$

Es decir, $P \triangleright R$ si satisface toda regla de suficiencia r_s y toda regla de necesidad r_n dentro de *R*.

Miklau y Suciu definen PS(R) como el conjunto de todas las *protecciones P* que satisfacen *R*:

$PS(R) = \{P \mid P = (t, \sigma),\ \forall\ r \in R, P \triangleright r\}$

De igual forma definen:

> "Dadas dos *protecciones P* y *P'* sobre el mismo árbol de meta-datos t_m, *P* es más restrictiva de *P'*, expresado como la relación de preorden *P«P'*, si para todos los nodos $i \in NODES(t_m)$, se mantiene la implicación lógica $\varphi_i \models \varphi'_i$ " [1], vale la pena recordar que φ_i denota la regla de necesidad del elemento i, como se expuso en la sección 2.3. La reflexividad y transitividad se cumplen para este tipo de relaciones, por ejemplo si se tienen tres protecciones *P*, *P'* y *P''* de modo que *P«P'* y *P'«P''*, entonces *P«P''* [3].

Entonces se dice que:

> "Para un conjunto de *protecciones* S, GLB(S) denota el límite inferior (*Greatest Lower Bound*) sobre *«*"[1].

Por lo tanto, el objetivo al construir una protección ahora se puede reescribir como "calcular el GLB para *R*". Formalmente:

P_R = GLB(PS(*R*)) siempre que $|PS(R)| \neq 0$

Vale la pena considerar, en adelante, los siguientes dos elementos de notación:

$\bigwedge$ S es la abreviación de una expresión lógica del estilo: $\sigma_1 \wedge \sigma_2 \wedge \sigma_3 \ldots$ y

$\bigvee$ S es la abreviación de una expresión lógica del estilo: $\sigma_1 \vee \sigma_2 \vee \sigma_3 \ldots$

para S = { $\sigma_1, \sigma_2, \sigma_3 \ldots$ } donde σ_i puede ser una regla o una expresión lógica sobre varias reglas.

Teorema 1 (teorema 5.3 de Miklau y Suciu [1])

Sea *t* un árbol de documento, R_S el conjunto de reglas primitivas de suficiencia y R_N el conjunto de reglas primitivas de necesidad sobre *t*. Entonces:

- Si GLB(PS($R_N \cup R_S$)) existe, entonces es igual a GLB(PS(R_S)).
- GLB(PS(R_S)) siempre existe, y es la protección definida de la siguiente manera: para cada nodo $i \in$ NODES(*t*) la *fórmula llave* σ_i está dada por:

  $$\sigma_i = \bigvee \{ \bigwedge S \mid \exists (S \rightarrow e) \in R_S, i \preceq e \}$$

  Es decir, la *fórmula llave* del nodo *i* es la disyunción de todas las expresiones S que son suficientes para acceder algún descendiente de *i* y la de el propio *i*.

- Si GLB(PS($R_N \cup R_S$)) existe sii el siguiente **Criterio de Consistencia** se satisface: para cada par de reglas $(S \rightarrow e) \in R_S$ y $(e' \rightarrow S') \in R_N$, si $e' \preceq e$ (es decir, si *e'* es ancestro o el propio *e*), entonces $S' \subseteq S$.

Miklau y Suciu proporcionan la prueba del teorema en [1]

Considerando todo lo anterior, el procedimiento que Miklau y Suciu proponen para generar la protección comienza por revisar el cumplimiento del criterio de consistencia sobre el conjunto de reglas primitivas R, declarándolas inconsistentes si no se cumple o protegiendo, en el caso contrario, cada nodo *i* con la disyunción de sus reglas de suficiencia y las de todos sus descendientes, es decir, protegiendo a *i* con $\bigvee (\bigwedge S_j)\ \forall$

S_j que aparezca en una regla $(S \rightarrow e)$ en la que *e* sea *i* o cualquiera de sus descendientes; la figura 5 muestra un ejemplo del proceso para una protección construida a partir de reglas atómicas de suficiencia.

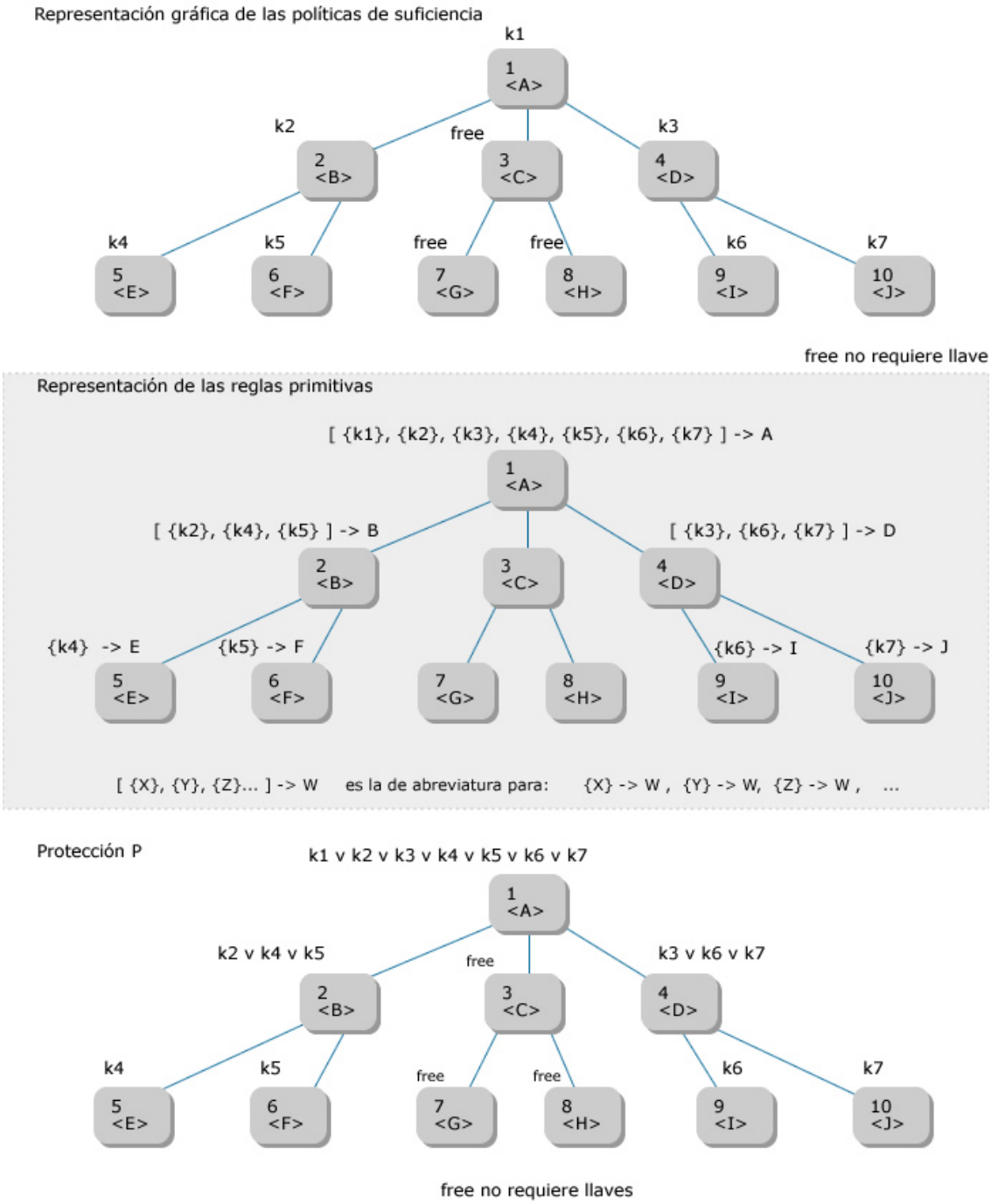

Figura 4: Políticas de acceso, reglas primitivas y protección

2.10.4 Normalización y cifrado del documento

Una vez generada la protección, es preciso normalizarla y cifrarla, la normalización se realiza sobre las *fórmulas llave* como se expuso en la sección 2.5 y el cifrado de acuerdo a lo presentado en la sección 2.6.1. Tras estos dos procesos, el documento puede ser publicado y sólo resta distribuir las llaves apropiadas para que los usuarios puedan consultarlo.

2.11 Conclusión

Hemos presentado cómo los documentos semiestructurados pueden ser representados como árboles y cómo los *tags* originales se pueden seleccionar, por medio de dos tipos de políticas de acceso, para ser controlados en su exposición mediante un conjunto de llaves criptográficas relacionadas entre sí de cierta manera. Presentamos, en la sección 2.3 y 2.10.3 las principales fórmulas que se construyen en el modelo de Miklau y Suciu a partir de las políticas de acceso sobre un documento; también expusimos, en la sección 2.6, la manera en la que Miklau y Suciu emplean meta-nodos para hacer posible, mediante procesos de cifrado real, el control de acceso sobre elementos específicos en el árbol. En resumen, hemos revisado, a lo largo del capítulo, todos los conceptos del modelo de Miklau y Suciu, desde los puramente teóricos en las secciones 2.1 a la 2.9 hasta el procedimiento para construir protecciones a partir cualquier documento, en la sección 2.10; dando por expuesto, con esto, el marco teórico que emplearemos en el resto del documento.