Preserving Confidentiality of Security Policies in Data Outsourcing

Sabrina De Capitani di Vimercati
DTI - Università di Milano
26013 Crema - Italy
decapita@dti.unimi.it

Sara Foresti
DTI - Università di Milano
26013 Crema - Italy
foresti@dti.unimi.it

Sushil Jajodia
CSIS - George Mason University
Fairfax, VA 22030-4444
jajodia@gmu.edu

Stefano Paraboschi
DIIMM - Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

Gerardo Pelosi
DIIMM - Università di Bergamo
24044 Dalmine - Italy
gerardo.pelosi@unibg.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

ABSTRACT

Recent approaches for protecting information in data outsourcing scenarios exploit the combined use of access control and cryptography. In this context, the number of keys to be distributed and managed by users can be maintained limited by using a public catalog of tokens that allow key derivation along a hierarchy. However, the public token catalog, by expressing the key derivation relationships, may leak information on the security policies (authorizations) enforced by the system, which the data owner may instead wish to maintain confidential.

In this paper, we present an approach to protect the privacy of the tokens published in the public catalog. Consistently with the data outsourcing scenario, our solution exploits the use of cryptography, by adding an encryption layer to the catalog. A complicating issue in this respect is that this new encryption layer should follow a derivation path that is "reversed" with respect to the key derivation. Our approach solves this problem by combining cryptography and transitive closure information. The result is an efficient solution allowing token release and traversal of the key derivation structure only to those users authorized to access the underlying resources. We also present experimental results that illustrate the behavior of our technique in large settings.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

WPES'08, October 27, 2008, Alexandria, Virginia, USA.

General Terms

Security, Management

Keywords

Data outsourcing, encryption policy, security policy protection, privacy

1. INTRODUCTION

The realization of access control policies that rely on cryptography to ensure confidentiality of data has traditionally been considered incoherent with the basic security principle of strict separation between policies and mechanisms. The evolution of Information and Communication Technologies (ICTs) is forcing a reconsideration of this attitude: systems present a growing variety of integrated components, with ever growing interconnection needs. The assumption that access to resources is controlled by an omniscient reference monitor exercising complete and efficient surveillance on every request is becoming increasingly impractical. There is therefore an increasing interest in the definition of security solutions that allow the enforcement of regulations even when the resources themselves are not under the strict control of the owner. A promising solution in this respect is represented by approaches coupling authorizations and encryption so to define encryption policies on resources that reflects the access regulations to be enforced. The problem of ensuring that users can only access resources for which they hold authorizations is then translated into guaranteeing that only users with access privileges are able to retrieve the key used to protect a resource.

One application that already shows a great potential from the adoption of these techniques is represented by the data outsourcing and dissemination services, which have recently seen considerable growth and promise to become a common component of the future Web. Users have access to an increasing variety of devices generating and processing digital information. User owned content is therefore more and more stored and managed by third parties delegated for this service by the owner. Many companies have recently experienced significant success in the Web space providing solutions that facilitate the dissemination of user-generated content (e.g., YouTube, Facebook). In all these scenarios, the server is relied upon for ensuring availability of outsourced data and for enforcing the basic security control on the data

it stores. While trustworthy with respect to their services in making published information available, third parties are however trusted neither to access the content nor to fully enforce possible access control policy reflecting selective release that the owner may wish to impose. It is then conceivable that users (and providers themselves) would find an interesting opportunity in the realization of a dissemination service offering strong guarantees about the protection of user privacy against the service provider. Recently, selective encryption techniques have emerged as a promising response to this problem [9, 10]. Selective encryption nicely enhances outsourcing solutions by safely delegating to the server itself also the management of the access control policy while ensuring complete protection of the content as well as correct enforcement of the policy against possible misbehaviors and collusions. To limit the number of keys to be assigned to users when applying selective encryption, the proposed approaches exploit the use of a public token catalog allowing key derivation along a hierarchy. The access control policy can then be safely and simply enforced by defining a key derivation hierarchy, a user key assignment, and a resource encryption policy in such a way that each user can, from her own key and the public tokens, derive all and only the keys enabling decryption of resources that the user is authorized to access.

The use of a key derivation hierarchy and its tokens, while greatly simplifying key management, introduces however a new vulnerability related to policy confidentiality. As a matter of fact, public availability of tokens, and therefore of the corresponding key derivation hierarchy, makes visible the relationship between users and resources they are authorized to access, and therefore discloses the authorization policy. In several contexts, however, the policy itself should be considered confidential as owners do not wish to publicly declare to whom they give (or not give) access to their resources. Also, an analysis of the policy may allow observers to reconstruct the structure of the social network of users accessing the system and of their real identities. Since the overall aim of these novel solutions is to allow an efficient confidentiality-preserving mechanism for resource dissemination, the protection of the access control policy appears a natural requirement that systems will be interested in supporting, as long as system performance remains guaranteed.

In this paper, we present a solution nicely complementing selective encryption by protecting the privacy of the security policy to be (indirectly) enforced by the third party. Consistently with the context to which it is applied, our solution basically provides a protection (encryption) layer to the policy information exploiting the key derivation hierarchy itself. Our approach effectively combines encryption and transitive closure information and takes into account the performance requirements, considering several parameters that characterize the system's behavior in this scenario. The resulting solution is shown to be efficient and promises to become a natural component of real-world services exploiting the features of the techniques developed in the last few years.

The remainder of this paper is organized as follows. Section 2 introduces the access control model we assumed used in the data outsourcing scenario, presenting also the public catalog necessary for allowing users to access data and interacting with the server. Section 3 presents our a solution for protecting the access control policy enforced by the system, while limiting the impact on the client-server interaction.

	$\mathbf{r_1}$	$\mathbf{r_2}$	$\mathbf{r_3}$	$\mathbf{r_4}$	$\mathbf{r_5}$
A	1	1	0	1	1
B	1	1	1	1	1
C	0	1	1	1	1
D	0	0	1	1	1

Figure 1: An example of access matrix

Section 4 presents experimental results. Section 5 describes related work. Finally, Section 6 presents our concluding remarks.

2. BASIC CONCEPTS

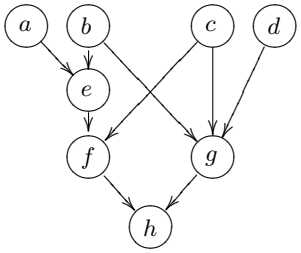
We consider a system with a set $\mathcal{U}$ of users, a set $\mathcal{R}$ of resources, and a policy regulating the accesses of users to resources (consistently with the data outsourcing scenario, we assume access to be read-only) modeled via an access matrix $\mathcal{A}$ with $|\mathcal{U}|$ rows and $|\mathcal{R}|$ columns. Each entry $\mathcal{A}[u, r]$ is set to 1 if u can access r; it is set to 0 otherwise. Given an access matrix $\mathcal{A}$, $acl(r)$ denotes the *access control list* of r (i.e., the set of users that can access r). Figure 1 illustrates a sample access matrix with four users (A, B, C, D) and five resources (r_1, ..., r_5), where, for example, $acl(r_2)$=$\{A,B,C\}$.

2.1 Encryption policy

Since the enforcement of the access control policy cannot be delegated to the remote server, which is not trusted for playing this role, we have proposed the application of a *selective encryption technique* [10]. This technique uses different keys for encrypting data and gives to each user a set of keys that allow her to decrypt all and only the resources she is authorized to access. The key distribution required by the selective encryption can be efficiently managed through a *key derivation strategy* via tokens [4]. The release to each user of a set $K = \{k_1, \ldots, k_n\}$ of keys can be obtained through the release to each user of a single key $k_i \in K$ and a set of *tokens* necessary to derive (directly or indirectly) all keys $k_j \in K$, with $j \neq i$. Given two keys k_i and k_j, a token $t_{i,j}$ is defined as $t_{i,j} = k_j \oplus H(k_i, l_j)$, where l_j is a publicly available label associated with k_j, $\oplus$ is the bitwise `xor` operator, and H is a deterministic cryptographic function. Key derivation via tokens can be applied in chains: a chain of tokens is a sequence $t_{i,l}, \ldots, t_{n,j}$ of tokens such that $t_{c,d}$ directly follows $t_{a,b}$ in the chain only if $b = c$.

Graphically, a set $\mathcal{K}$ of keys and a set $\mathcal{T}$ of tokens can be represented via a *key derivation graph*, having a vertex v_i for each key $k_{v_i} \in \mathcal{K}$, and an arc (v_i, v_j) for each token $t_{v_i,v_j} \in \mathcal{T}$. Chains of tokens correspond then to paths in the graph. For simplicity, we assume that each vertex v_i is uniquely identified by the public label associated with the corresponding key k_{v_i}, that is, v_i=l_i. A key assignment function ϕ determines the label $\phi(u)$ of the vertex corresponding to the key assigned to u. Analogously, the key assignment function determines the label $\phi(r)$ of the vertex corresponding to the key with which r is encrypted.

The specification of keys and tokens must guarantee that each user can, from her own key and the tokens publicly available, derive all and only the keys used to encrypt the resources she is authorized to access. A straightforward approach to keys and token specification that satisfies this property consists in: *i)* creating a key for each access control list in $\mathcal{A}$ and for each singleton set of users that is not already included in the *acl*s in $\mathcal{A}$; *ii)* define a token between pairs of keys corresponding to sets of users for which there is a *direct*



(a)

u	$\phi(u)$
A	a
B	b
C	c
D	d

(b)

r	$\phi(r)$
r_1	e
r_2	f
r_3	g
r_4,r_5	h

(c)

Figure 2: An example of key derivation graph (a) and key assignment (b)-(c)

LABELS

res_id	label
r_1	e
r_2	f
r_3	g
r_4	h
r_5	h

TOKENS

token_id	source	destination	token_value
α	a	e	$k_e \oplus H(k_a,e)$
β	b	e	$k_e \oplus H(k_b,e)$
γ	b	g	$k_g \oplus H(k_b,g)$
δ	c	f	$k_f \oplus H(k_c,f)$
ε	c	g	$k_g \oplus H(k_c,g)$
ζ	d	g	$k_g \oplus H(k_d,g)$
η	e	f	$k_f \oplus H(k_e,f)$
θ	f	h	$k_h \oplus H(k_f,h)$
ι	g	h	$k_h \oplus H(k_g,h)$

Figure 3: Token catalog for Example 2.1

containment relationship [10]; *iii)* assign each user u to the key corresponding to the singleton set $\{u\}$ and encrypt each resource r with the key corresponding to $acl(r)$.

EXAMPLE 2.1. *Consider the access matrix in Figure 1. Since there are 4 different acls (i.e., $\{A,B\}$, $\{A,B,C\}$, $\{B,C,D\}$, $\{A,B,C,D\}$), and 4 users we define 8 different keys, one for each acl and one for each user. In particular, keys $k_{\mathrm{a}},\ldots,k_{\mathrm{d}}$ are associated with users $A,\ldots,D$, respectively, k_{e} is associated with $\{A,B\}$, k_{f} with $\{A,B,C\}$, k_{g} with $\{B,C,D\}$, and k_{h} with $\{A,B,C,D\}$. We then define a token between each pair of keys (k_x,k_y), $x,y \in \{\mathrm{a},\mathrm{b},\ldots,\mathrm{h}\}$ and $x \neq y$, such that the set of users corresponding to k_x is a subset of the set of users corresponding to k_y. Figure 2(a) illustrates the graphical representation of these keys and tokens, where vertex v corresponds to key k_v. Figures 2(b) and 2(c) illustrate the key assignment function ϕ enforcing the policy.*

2.2 Token management

The set $\mathcal{T}$ of tokens computed on the set $\mathcal{K}$ of keys in the system is made *publicly available* by storing a *token catalog* on the server [9, 10]. The token catalog is composed of two tables: LABELS and TOKENS. Table LABELS maintains the correspondence between a resource (attribute *res_id*) and its vertex label (attribute *label*). TOKENS contains a tuple for each token $t_{v_i,v_j} \in \mathcal{T}$ and is characterized by four attributes: *token_id* is the token identifier; *source* and *destination* are the labels of the corresponding source and destination vertices in the graph; *token_value* is the token value computed as $k_{destination} \oplus H(k_{source},destination)$. Figure 3 illustrates tables LABELS and TOKENS corresponding to the key derivation graph and the key assignment function in Figure 2.

Consider user u who needs to access resource r. Figure 4 describes the algorithm that receives as input the resource identifier r, the key $k_{\phi(u)}$ known to u, and the label $\phi(u)$,

INPUT
$\phi(u)$: label of the user's key
$k_{\phi(u)}$: user's key
r: resource to be accessed
OUTPUT
$k_{\phi(r)}$: key with which r is encrypted

KEY_DERIVATION
1. $chain$:= **Find_Path**($\phi(u)$,r) /* server-side query */
2. **if** $chain \neq \emptyset$ **then** /* client-side computation */
   t := **Pop**($chain$)
   repeat
   $k_{t[destination]}$:= $t[token\_value] \oplus H(k_{t[source]}, t[destination])$
   t := **Pop**($chain$)
   until t=NULL
   return($k_{t[destination]}$)
  return($\emptyset$)

FIND_PATH($from$,r)
Let $t \in$ LABELS | $t[res\_id]=r$
to := $t[label]$
Topologically sort V in $\mathcal{G}$
for each $v \in V$ **do**
   $dist[v]$:= ∞
   $pred[v]$:= NULL
$dist[from]$:= 0
for each $v_i \in V$ **do** /* visit vertices in topological order */
   for each $(v_i,v_j) \in A$ **do** /* the weight of each arc is 1 */
      if $dist[v_j] > dist[v_i]+1$ **then**
         $dist[v_j]$:= $dist[v_i]+1$
         $pred[v_j]$:= v_i
$chain$:= $\emptyset$
$current$:= to
while $current \neq from$ AND $current \neq$NULL **do**
   Let $t \in$ TOKENS | $t[source]=pred[current]$
                AND $t[destination]=current$
   Push($chain$,t)
   $current$:= $pred[current]$
if $current$=NULL **then return**($\emptyset$)
else return($chain$)

Figure 4: Key derivation

and computes the key $k_{\phi(r)}$ with which resource r is encrypted. The algorithm is composed of two steps. The first step is based on function **Find_Path** operating server-side that, given a label $\phi(u)$ and a resource r, retrieves the shortest token chain from $\phi(u)$ to $\phi(r)$. Basically, function **Find_Path** first determines $\phi(r)$ by querying table LABELS and then computes the shortest path in the key derivation graph. To this aim, it uses a shortest path algorithm (an improved version of Dijkstra working on DAGs), which exploits the topological order of vertices. Function **Find_Path** then starts from vertex $current$=$\phi(r)$ and builds backward the path to $\phi(u)$, following at each step $pred[current]$, which is an array that contains the label of the predecessor of vertex $current$ in the path previously computed, and adds to $chain$ the token in TOKENS from $pred[current]$ to $current$. The second step is evaluated client-side and consists in deriving keys following the chain of tokens (if not empty) returned by **Find_Path**, thus terminating with the derivation of $k_{\phi(r)}$.

EXAMPLE 2.2. *Consider the key derivation graph in Figure 2(a) and the catalog in Figure 3 and suppose that B, with $\phi(B)$=b, wants to access r_4. Function* **Find_Path**(b,r_4) *computes $\phi(r_4)$=h and finds the shortest path in the key derivation graph from* b *to* h*, thus setting* pred[h] *to* g *and* pred[g] *to* b*. The returned* chain *is then composed of two tokens corresponding to the tuples of table* TOKENS *with* token_id *ι and γ. The algorithm derives k_{g} through user's secret key k_{b} and token γ; it derives k_{h} (which corresponds to $k_{\phi(r_4)}$) through the just computed k_{g} and token ι.*

3. PROTECTION OF THE ACCESS CONTROL POLICY

A drawback of the public availability of tables TOKENS and LABELS is that users can infer the access control policy defined by the data owner. Indeed, table TOKENS completely describes the topology of the key derivation graph, and table LABELS precisely indicates which resource is associated with each of the vertices mentioned in TOKENS. This implies that any subject accessing the server could retrieve the policy adopted by the owner. Analogously, any authorized user of the system would be able to infer the existence of other users in the system, authorized to read a set of resources possibly wider than hers. To solve this problem, we propose an encryption strategy that, while preserving efficiency in key derivation, allows each user u to access only the portion of the key derivation graph that she is authorized to know, that is, the sub-graph rooted at the vertex labeled $\phi(u)$.

3.1 Protection with encryption and no additional information

A simple approach for protecting the topology of the key derivation graph consists in encrypting the tokens stored in table TOKENS in such a way to preserve the ability of each user to retrieve the tokens needed to derive the keys necessary to decrypt the resources that she can access. To this purpose, given a tuple t of table TOKENS (i.e., a token), the key associated with $t[source]$ is used for encrypting $t[destination]$ and $t[token\_value]$ as a whole. In this way, we have the guarantee that the decryption operation can be performed only by users that directly or indirectly are authorized to know the key associated with $t[source]$. The advantage of this solution is that the topology of the key derivation graph (and therefore the access control policy) is protected since, from the encrypted tokens, a user can only infer the number of vertices in the graph and the number of the outgoing arcs of each vertex but cannot infer anything about their connections. The drawback of this approach is that the process for deriving a specific key becomes expensive since, in the worst case, a user needs to traverse the whole sub-graph rooted at the vertex corresponding to her key. Indeed, to derive the key associated with a particular vertex, which we call *target_vertex*, a user u can only perform a top down traversal of the key derivation graph, starting from the vertex corresponding to $\phi(u)$. The user therefore should interact with the server to progressively retrieve the tokens that allow the derivation of the keys associated with the descendant vertices of $\phi(u)$, until she reaches the key of interest or the visit terminates.

EXAMPLE 3.1. *Suppose that user C wants to access resource r_4. According to table* LABELS*, the target vertex is* h*. User C knows that her key is associated with vertex* c *and therefore she first retrieves from the server the encrypted tokens with source equal to* c*. The returned tokens (δ and ε) correspond to the outgoing arcs of vertex* c*. After their decryption, C is able to see that the destinations of such tokens are vertices* f *and* g *and therefore can derive their keys. Subsequently, C retrieves the tokens with source equal to* f *(token θ) or* g *(ι), decrypts them by using the key derived in the previous step, and computes the key associated with vertex* h*, which correspond to their destination.*

As it is visible from this example, the increase in the computational time of the key derivation process is due to the fact that the user has no information on the position of the target vertex within the key derivation graph. Consequently, the user can only blindly explore the sub-graph rooted at her vertex labeled $\phi(u)$. The fundamental observation for efficiently reaching the target vertex is that the key derivation process can be seen as a reachability query on the key derivation graph. In fact, to check whether a given user whose key is associated with vertex v_i can derive the key associated with target vertex v_j, it is necessary to check whether vertex v_j is reachable from vertex v_i through a path in the graph. Our goal is then to develop a strategy for efficiently solving reachability queries and for efficiently retrieving a path in the key derivation graph.

3.2 Computation of reachability information

In the literature there are different approaches that can be used for solving a reachability query; we chose to carefully adapt the technique presented by Agrawal et al. [2] because it provides optimality guarantees on the size of the additional information. Given a DAG, the technique in [2] labels the vertices using numerical intervals reflecting the transitive closure of the ancestor-descendant relationship. By looking at the intervals associated with a specific vertex v_i, it is then immediate to verify whether there is a path from vertex v_i to another vertex v_j. These intervals are computed in three steps. First, a spanning tree of the original graph that minimizes the number of intervals is determined. Second, each vertex in the spanning tree is associated with a *numeric identifier*, representing the position of the vertex in the postorder visit of the spanning tree. Each vertex v_i is also associated with an interval $I = [i_1, i_2]$, where i_2 is the numeric identifier associated with v_i and i_1 is the smallest numeric identifier of its descendants. Third, all vertices v_i in the original graph are examined in a reverse topological order. For every arc (v_i, v_j) in the DAG, the intervals associated with vertex v_j are added to the intervals of v_i.

While the approach presented in [2] minimizes the storage required for representing the compressed transitive closure relationship, it does not provide any guarantee on the length of the path that needs to be traversed for reaching a specific vertex, since its main goal is to efficiently represent the reachability property of the graph. In our context, the length of the path connecting two vertices is instead an important property that has to be minimized to provide an efficient key derivation process (as the longer the path the more the queries to be processed). It is reasonable that the number of queries on the server will represent in most scenarios the most important parameter limiting the performance in the retrieval of the keys by the user. We therefore present a variation of the technique proposed in [2] that permits the computation of the intervals in such a way to remove those associated with "redundant paths" and that guarantees the traversal of the shortest path between two connected vertices of the graph, thus supporting access to the catalog employing the minimum number of queries to the server.

3.3 Transitive closure materialization

At a high level our algorithm is composed of five steps. The first three steps correspond to the three steps of the Agrawal et al.'s proposal. In the fourth step, the intervals

INPUT
$\mathcal{G}(V,A)$: key derivation graph
OUTPUT
$\mathcal{G}(V,A)$: with $v.id\ \forall v \in V$ and $a.intervals\ \forall a \in A$

MAIN
1. /* assign id to each vertex in V */
  Introduce a root vertex $\top$ in $\mathcal{G}$
  Let $ST(V,A')$ be the depth-first spanning tree of $\mathcal{G}$
  $id := 1$
  Assign_Id($\top$)

2. /* assign *intervals* to each vertex in V */
  Find_Interval($\top$)

3. /* complete intervals considering also the additional arcs in $\mathcal{G}$ */
  for each $(\top,v)\in A$ **do** $A := A - \{(\top,v)\}$
  $V := V - \{\top\}$
  for each $v\in V$ **do** /* visit $\mathcal{G}$ in reverse topological order */
    for each $(v,v_i)\in A$ **do**
      for each $I\in v_i.intervals$ **do**
        if $\nexists\ I' \in v.intervals \mid I\subseteq I'$ **then**
          $v.intervals := v.intervals \cup \{I\}$

4. /* assign *intervals* to each arc in A */
  for each $a=(v_i,v_j)\in A$ **do** $a.intervals := v_j.intervals$

5. /* optimize arcs' *intervals* by removing redundancies */
  for each $v\in V$ **do**
    for each $a_i=(v,v_i)\in A$ **do**
      for each $a_j=(v,v_j)\in A \mid v_i\neq v_j$ **do**
        for each $id \mid (\exists\ I_i\in a_i.intervals \mid id\in I_i)$
            AND $(\exists\ I_j\in a_j.intervals \mid id\in I_j)$ **do**
          if Shortest_Path_Length($v_i.id,id$) $\leq$
            Shortest_Path_Length($v_j.id,id$) **then**
            Remove(id,a_j)
          else Remove(id,a_i)

ASSIGN_ID(v)
for each $(v,v_i)\in A'$ **do Assign_Id**(v_i)
$v.id := id$
$id := id+1$

FIND_INTERVAL(v)
$min := v.id$
for each $(v,v_i)\in A'$ **do**
  $idchild :=$ **Find_Interval**(v_i)
  if $idchild < min$ **then** $min := idchild$
$v.intervals := [min,\ v.id]$
return(min)

REMOVE(id,a)
Let $I=[i_1,i_2]\in a.intervals$ such that $id\in I$
case id **of**
  i_1: $a.intervals := a.intervals - \{I\} \cup \{[id+1,i_2]\}$
  i_2: $a.intervals := a.intervals - \{I\} \cup \{[i_1,id-1]\}$
  default: $a.intervals := a.intervals - \{I\} \cup \{[i_1,id-1],[id+1,i_2]\}$

Figure 5: Derivation paths computation and materialization

associated with vertices are moved to the incoming arcs of the vertices. The advantage of having intervals associated with arcs (i.e., tokens) instead of having intervals associated with vertices is that the intervals can be immediately integrated within the encrypted version of the tokens, thus increasing the efficiency of the key derivation process without impacting the storage requirements (see Section 4). Intervals on vertices would instead require a new table and a doubling of the number of queries to the server. In the fifth step, the possible *redundancies* among intervals are removed. A redundancy may happen when there are two or more paths connecting vertex v_i to vertex v_j. In this case, there are two or more outgoing arcs of vertex v_i such that their intervals contain the numeric identifier associated with vertex v_j. We therefore modify such intervals so that the numeric identifier associated with v_j is included only within the interval associated with arcs that belong to the shortest path between v_i and v_j.

The algorithm for computing and materializing a transitive closure for a given key derivation graph is represented in Figure 5. The algorithm takes as input a key derivation graph $\mathcal{G}(V,A)$, computes its transitive closure, and returns a labeled version of the graph, where each vertex $v\in V$ is associated with an identifier $v.id$ and each arc $a\in A$ is associated with a set of intervals $a.intervals$, describing the identifiers of the vertices reachable along that arc. The algorithm is structured in the following 5 steps.

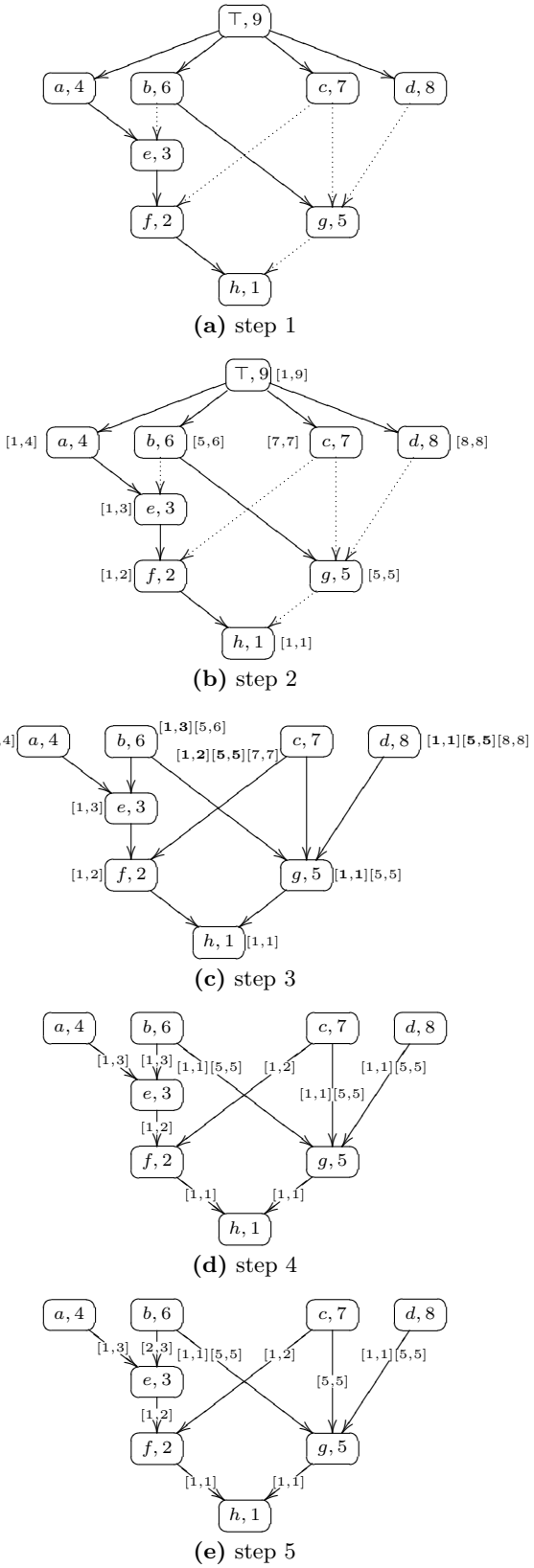
1. The algorithm first extends $\mathcal{G}$ with a root vertex $\top$, then it finds the depth-first spanning tree ST of the key derivation graph $\mathcal{G}$, and calls procedure **Assign_Id**, which recursively executes a postorder visit of ST. For each visited vertex v, the procedure associates a progressive numeric identifier $v.id$, reflecting the order in which vertices have been visited. Therefore, the identifier of the left-most leaf in ST is 1, while the identifier of the root is n, where n is the number of vertices in V. Note that, by construction, each vertex v has an identifier $v.id$ higher than that of its ancestors in the ST and all its ancestors have contiguous identifiers.

2. The algorithm calls function **Find_Interval** that assigns to each vertex v in V an interval $v.intervals$ representing the set of vertices reachable from v, following only arcs belonging to spanning tree ST. Function **Find_Interval** recursively determines, for each vertex v, the minimum id (represented by variable min) in the subtree rooted at v. The interval associated with v is then $[min, v.id]$, meaning that all the vertices with an identifier between min and $v.id$ belong to the subtree of ST rooted at v.

3. The algorithm removes from the graph the vertex $\top$ as well as all its outgoing arcs. Then, the algorithm completes the *intervals* information associated with each vertex to represent the reachability property of the whole graph, by visiting $\mathcal{G}$ in reverse topological order. For each vertex v, the algorithm checks all arcs (v,v_i) in the set A of arcs in $\mathcal{G}$. It then adds to $v.intervals$ all intervals in $v_i.intervals$ that are not already contained in an interval in $v.intervals$ (an interval $I=[i_1,i_2]$ is contained in another interval $I'=[i'_1,i'_2]$, denoted $I\subseteq I'$, only if $i'_1 \leq i_1$ and $i_2 \leq i'_2$). The reverse topological ordering ensures that intervals are assigned to a vertex only after those of its descendants have been calculated.

4. Intervals are passed from vertices to their incoming arcs. Therefore, for each arc $a=(v_i,v_j)\in A$, $v_j.intervals$ is assigned to $a.intervals$. In this way, it is always possible to easily determine the set of vertices that can be reached through a path starting at v_i.

5. The algorithm removes redundancies from intervals. Given a vertex v_i, and a descendant v_j of it in the graph, there may exist different paths connecting v_i to v_j. Since it is not necessary to keep track of all these paths, we maintain the shortest path only. For each pair of arcs a_i, a_j starting at the same vertex v, and for each id belonging to both an interval in $a_i.intervals$



(a) step 1

(b) step 2

(c) step 3

(d) step 4

(e) step 5

Figure 6: An example of the execution of algorithm in Figure 5

LABELS

res_id	label
r_1	e
r_2	f
r_3	g
r_4	h
r_5	h

IDS

label	vertex_id
a	4
b	6
c	7
d	8
e	3
f	2
g	5
h	1

ENCTOKENS

token_id	source	enc_token
α	a	$\mathrm{E}_{k_a}(e,\ k_e \oplus H(k_a,e),\ [1,3])$
β	b	$\mathrm{E}_{k_b}(e,\ k_e \oplus H(k_b,e),\ [2,3])$
γ	b	$\mathrm{E}_{k_b}(g,\ k_g \oplus H(k_b,g),\ [1,1][5,5])$
δ	c	$\mathrm{E}_{k_c}(f,\ k_f \oplus H(k_c,f),\ [1,2])$
ε	c	$\mathrm{E}_{k_c}(g,\ k_g \oplus H(k_c,g),\ [5,5])$
ζ	d	$\mathrm{E}_{k_d}(g,\ k_g \oplus H(k_d,g),\ [1,1][5,5])$
η	e	$\mathrm{E}_{k_e}(f,\ k_f \oplus H(k_e,f),\ [1,2])$
θ	f	$\mathrm{E}_{k_f}(h,\ k_h \oplus H(k_f,h),\ [1,1])$
ι	g	$\mathrm{E}_{k_g}(h,\ k_h \oplus H(k_g,h),\ [1,1])$

Figure 7: Encrypted catalog for the example in Figure 3

and an interval in $a_j.intervals$, the algorithm computes the length of the shortest paths reaching id from v and passing through a_i and a_j, respectively. Identifier id is then removed from the intervals associated with the arc involved in the longest of the two paths, possibly splitting the interval containing id (see function **Remove** in Figure 5). If the paths have the same length, id is however removed from one of the arcs to avoid redundancy.

EXAMPLE 3.2. *Figure 6 presents the execution step by step of the algorithm in Figure 5 applied to the example in Figure 2. First, the algorithm adds a virtual root labeled* $\top$ *to the graph in Figure 2*(a) *and builds the depth-first spanning tree ST on* $\mathcal{G}$*. Trees 6*(a)(b) *in Figure 6 represent* $\mathcal{G}$ *after the execution of step* 1 *and step* 2*, respectively. Figure 6*(c) *represents* $\mathcal{G}$ *after the execution of step* 3*, where vertex* $\top$ *together with its outgoing arcs are removed from* $\mathcal{G}$*. Figure 6*(d) *represents* $\mathcal{G}$*, where the intervals have been moved to arcs. Finally, Figure 6*(e) *represents the labeled graph returned by the algorithm, where redundant labels have been removed. To illustrate, consider vertices* b *and* h *(with* id 1*). They are connected by two different paths, but the algorithm maintains only the one passing through* e*, since it is the shortest.*

3.4 Confidential policy management

After the computation of the transitive closure of the key derivation graph, the information about the encrypted tokens is stored in three tables: LABELS, IDS, and ENCTOKENS. As in [10], table LABELS maintains the correspondence between a resource (attribute *res_id*) and its vertex label (attribute *label*). Table IDS maintains the correspondence between a vertex label (attribute *label*) and its numeric identifier (attribute *vertex_id*) computed in the first step of the algorithm in Figure 5. Table ENCTOKENS has the same role as table TOKENS in [10] but its content is partially encrypted. More precisely, table ENCTOKENS contains a tuple for each token t_{v_i,v_j} and is charac-

INPUT
$\phi(u)$: label of the user's key
$k_{\phi(u)}$: user's key
r: resource to be accessed

OUTPUT
$k_{\phi(r)}$: key necessary to decrypt r

MAIN
1. Let $t \in$ LABELS $\mid t[res\_id]=r$ /* server-side query */
   $target\_label := t[label]$
2. Let $t \in$ IDS $\mid t[label]=target\_label$ /* server-side query */
   $target\_id := t[vertex\_id]$
3. $current := \phi(u)$
   while $current \neq target\_label$ **do**
     /* server-side query */
     $token\_set := \{t \in$ ENCTOKENS$\mid t[source]=current\}$
     if $token\_set=\emptyset$ **then return**(NULL)
     $found :=$ FALSE
     while ($found=$FALSE) AND ($token\_set \neq \emptyset$) **do**
       Let $t \in token\_set$
       $token\_set := token\_set - \{t\}$
       $[destination, token\_value, intervals] := D_{k_{t[source]}}(t[enc\_token])$
       if $target\_id \in intervals$ **then**
         $k_{destination} := token\_value \oplus H(k_{t[source]}, destination)$
         $current := destination$
         $found :=$ TRUE
     if $found=$FALSE **then return**(NULL)
   return(k_{target_label})

Figure 8: Key derivation on the encrypted catalog

terized by three attributes: *token_id* is the token identifier; *source* is the label of the source vertex of the token; and *enc_token* represents the encryption portion of the token and is obtained by encrypting with k_{source} the concatenation of the destination of the token, the token value computed as $k_{destination} \oplus H(k_{source}, destination)$, and the intervals associated with the token and computed by the algorithm in Figure 5.

Figure 7 illustrates table ENCTOKEN corresponding to table TOKEN in Figure 3, considering the intervals represented in Figure 6(e).

Figure 8 illustrates the key derivation process on the encrypted catalog like in [10] (see Section 2.2). The algorithm receives as input a resource identifier r, the key $k_{\phi(u)}$ known by user u, and the corresponding label $\phi(u)$, and returns the key $k_{\phi(r)}$ with which resource r is encrypted. The algorithm first determines $\phi(r)$ and the corresponding vertex identifier (variable *target_id*) by querying tables LABELS and IDS on the remote server. Then, it visits the key derivation graph starting from vertex $\phi(u)$. At each iteration of the external **while** loop, the algorithm retrieves from table ENCTOKENS the set *token_set* of tuples representing the outgoing arcs of the *current* vertex, that is, the tokens whose source is the *current* vertex. At each iteration of the internal **while** loop, attribute $t[enc\_token]$ of a tuple t in *token_set* is decrypted via key $k_{t[source]}$, which has been computed in the previous iteration of the external **while** loop or is already known to the user. From such an operation, the algorithm retrieves the *destination*, the *token_value*, and the *intervals* of the current token. If the *target_id* belongs to an interval in *intervals*, the new *current* vertex is set to *destination*; otherwise, another tuple in *token_set* is analyzed. The algorithm terminates when either $\phi(r)$ is reached (i.e., *current* = *target_label*) or when $\phi(r)$ is not reachable from $\phi(u)$ (in the case u is not authorized to read r and cannot therefore retrieve its key).

EXAMPLE 3.3. *Consider the catalog in Figure 7 and suppose that B, with* $\phi(B)$=b, *wants to access* r_4. *The algorithm retrieves* target_label=$\phi(r_4)$=h, *the corresponding* target_id=*1, and sets* current *to* b. *Then, it retrieves from table* ENCTOKENS *the tuples with attribute source equal to* b, *that is, tuples with* token_id β *and* γ, *and decrypts them. Since* target_id *belongs only to* intervals *associated with token* γ, *whose destination is* g, *the algorithm computes* k_g *and sets* current *to* g. *The algorithm then retrieves all the tuples in table* ENCTOKENS *with* source *equal to* g, *that is, tuple with* token_id ι. *After decrypting the token, the algorithm computes* k_h *and* current *is set to* h. *Since* $\phi(r_4)$=h, *the algorithm terminates.*

A concern may arise that the content in tables IDS and ENCTOKENS may still leak information on the topology of the key derivation graph, and therefore on the policy. We can observe three different sources of exposure: (1) the numerical identifiers in table IDS reflect the ordering in the postorder visit of the graph, and therefore may leak information on the topology of the graph; (2) the number of tokens corresponding to a given label leak the number of outgoing arcs characterizing each vertex; (3) the values of attribute *enc_token* have a variable size that depends on the number of intervals described in each token. The most significant leakage relates to the combination of the first two sources; however it can be easily proved that, assuming an upper limit on the number of direct descendants each vertex can have, different graphs (their number grows exponentially with the number of vertices) may correspond to the same visit order and number of descendants. The three kinds of leakage can then be considered negligible and we do not discuss them further. We note however that trivial countermeasures can be applied to block each of them, respectively: (1) numerical identifiers in IDS can start from a random value and then cycle back, or fake entries can be added at the start and at the end of the interval, to hide the initial and final point of the postorder visit; (2) fake tokens can be added to table ENCTOKENS to keep the number of outgoing arcs constant for all the vertices; (3) padding can be used to make the size of attribute *enc_token* uniform, with no significant impact on storage requirements.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The algorithms presented in this paper were all implemented, and experiments have been performed to assess effectiveness and performance of the proposed solution. The experiments demonstrate that our technique quickly produces the labeling and the intervals for the optimal reconstruction of the path. The comparison between the different algorithms shows the advantages deriving from the use of our approach.

4.1 Experimental setting

As representative of a potential selective dissemination scenario, we consider the case study, also analyzed in [8], of a sport news database.

The chosen service manages a system with t *teams*, where each *team* is composed by *pt players* and is coordinated by one manager. The service is supposed to be used by s team supporters, referred in the following as *subscribers*. Moreover, a set of *reporters* follows the league and uses the service

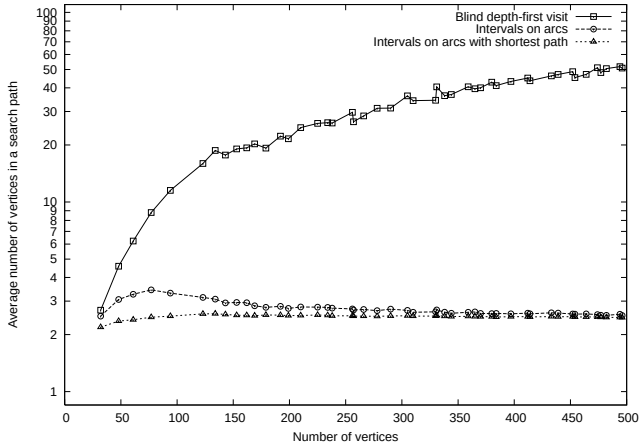


Figure 9: Average number of vertices traversed during a search operation

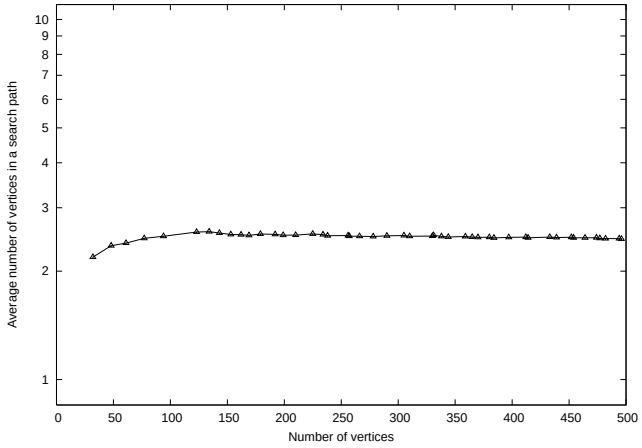


Figure 10: Average number of vertices traversed in a search operation via the shortest-path method

to work with tr teams. The *reporters* are grouped into sets of rm elements, each of which coordinated by one *manager*.

In the considered scenario, each subject (team manager, reporter, reporter manager, and subscribers) can subscribe to any number of resources, partitioned between *player news* and *team news*. Consistently with [8], the set of authorizations granted to subscribers is modeled to be quite large to evaluate the algorithms in a significant scenario. The number of team news accessed by each subscriber, along with the player news of the same team, follows a *Zipf* distribution that increases with the number s of subscribers.

We ran experiments by varying the number of teams from 2 to 50 and by considering the case of $s = 10$ subscribers, $pt = 5$ players per team, $tr = 5$ teams per reporter and $rm = 5$ reporters per manager. The key derivation graphs corresponding to the above settings show a number of vertices in the range [30, 500], where each vertex represents a distinct configuration of access control list.

4.2 Performance evaluation

Our goal is to describe the performance of the technique proposed in the paper along three directions. The first analyzes the performance in terms of the number of vertices visited on the graph compared to the number of vertices visited by a blind search that does not use intervals on the arcs. The second estimates the average number of vertices that have to be visited to reach, starting from a user vertex, all the vertices in the graph. The third evaluates the storage requirements necessary to materialize the whole transitive closure, comparing the technique in [2], which associates intervals with the vertices, and our approach, which stores intervals on the arcs and keeps only those representing the shortest path.

The behavior is shown in Figures 9–11, where separate metrics are used, all expressed as a function of the number of vertices of the input graph, using a logarithmic scale on the y-axis to better visualize percentage variations (a constant percentage gain corresponds to a constant linear distance on the y-axis).

Figure 9 shows the average length of a token chain resulting from the retrieval of a target key. The value on the y-axis was computed averaging the number of accessed vertices resulting from the search of every key in the graph. As expected, the presence of the transitive closure information, in the form of intervals on the tokens, produces a configuration able to outperform a blind depth-first search by a factor greater than 10 for all graphs having more than 100 vertices. The comparison between the algorithm that uses the interval data stored on arcs (i.e., the graph obtained at the end of Step 4 of the algorithm in Figure 5) and the one that minimizes the token chains using the shortest-path approach (Step 5 of algorithm in Figure 5) outlines a better performance of our proposal, with a percentage gain which reaches a value of 36%.

Figure 10 plots the average number of vertices traversed during a search operation by our solution (i.e., the graph as obtained at the end of Step 5 of the algorithm), considering the average on the search of all the vertices in the graph. The number of visited vertices corresponds to the number of queries on table TOKENS that are needed to obtain the information required to extract the target key. The graphic confirms that the number of vertices traversed to reach a target key is low (stable at around 2.5). The value does not grow with the increase in the size of the graph, providing convincing evidence that the technique for policy protection proposed in the paper is able to provide an efficient retrieval of keys even for large configurations.

Since the number of arcs in a graph grows potentially as $O(n^2)$, where n is the number of vertices, it is important to verify that the technique does not force excessive storage requirements. Figure 11 shows that the proposed labeling procedure requires a total number of intervals on the arcs that is comparable (indeed, typically a little smaller) to the number of intervals on the vertices produced by the method of Agrawal et al. [2]. The experiments therefore confirm that our technique is manageable, even for large graphs and scenarios with large user populations and complex protection requirements.

5. RELATED WORK

Previous related work is in the area of database-as-a-service [12,14], which considers the problem of database outsourcing. Most of this research focuses on the design of indexing techniques that allow an efficient execution of queries

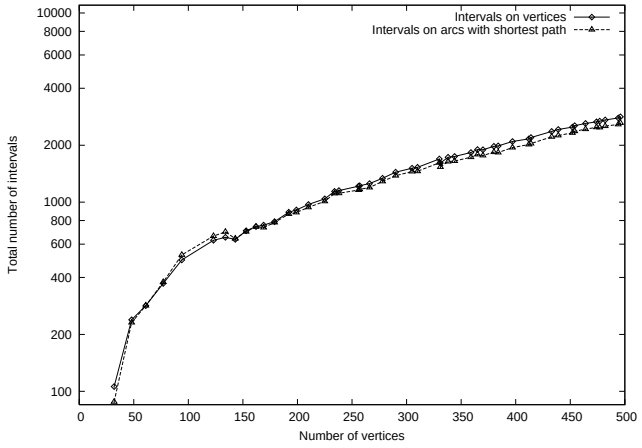


Figure 11: Total number of intervals using different labeling choices

(e.g., [12,14,15,22,23]). Although many indexing techniques that support various types of queries in the database-as-a-service scenario have been developed, few proposals have provided a deep analysis of the level of protection provided by all these techniques against inference and linking attacks. In [6] the authors provide an evaluation demonstrating that even a limited number of indexes can greatly facilitate an attacker who wants to violate the confidentiality provided by encryption.

Few research efforts have addressed the issues of access control in a data outsourced scenario. In [9, 10] the authors present a selective encryption strategy as as a means to enforce authorizations. Two layers of encryption are imposed on data: the inner layer is imposed by the owner for providing initial protection, the outer layer is imposed by the server to reflect policy modifications (i.e., grant/revoke of authorizations). We note that the approach proposed in this paper for preserving the confidentiality of security policies can be easily integrated with the model in [9, 10]. In [19] the authors propose a framework for enforcing access control on published XML documents by using different cryptographic keys over different portions of the XML tree and by introducing special metadata nodes in the structure. Some approaches for ensuring the integrity in the database outsourcing scenario have been also investigated [13, 20, 21]. All these approaches introduce the use of signatures combined with Merkle hash trees. Other proposals have studied solutions exploiting the combination of fragmentation and encryption for storing data on a single server by minimizing the amount of encrypted data, thus allowing for an efficient query execution [1, 7].

The problem of ensuring the confidentiality of access control policies has been investigated in the context of open environments [5,11,16,24,25]. These works are based on the assumption that parties may be unknown a-priori and therefore a multi-step trust negotiation process is necessary for communicating policies and for releasing certificates, which are both considered sensitive. Indeed, the goal of a trust negotiation process is to gradually establish trust among the parties, by disclosing credentials and requests for credentials. In the literature, a number of trust negotiation strategies has been proposed. PRUdent NEgotiation Strategy (PRUNES) [27] ensures that the client communicates her credentials to the server only if the access will be granted and the set of certificates communicated to the server is the minimal necessary for granting it. The Disclosure Tree Strategy (DTS) family of strategies [28, 29] is instead a closed set of strategies that grants interoperability, that is, if two parties use different strategies from the DST family, they are able to negotiate trust. TrustBuilder [26] is a prototype developed to incorporate trust negotiation into standard network technologies. Traust [18] is a third-party authorization service that is based on the TrustBuilder framework for trust negotiation. The Traust service provides a negotiation-based mechanism that allows qualified users to obtain the credentials necessary to access resources provided by the involved server.

A related line of research has studied the materialization of the transitive closure of graphs [2, 3, 17]. This topic has been widely studied to the aim of efficiently solving the graph reachability problem.

6. CONCLUSIONS

A crucial long-term objective of ICT research is represented by the investigation of the privacy issues that arise in the electronic society. The benefits, in terms of availability of large amounts of information and pervasive communication channels, have to be realized trying to minimize the creation of novel threats to the privacy of citizens. These privacy requirements are today not satisfied; service providers support large user communities, and their information interchange needs, in a way that gives rise to significant privacy violations.

Research on data outsourcing offers one of the most promising approaches for the realization of these services in a privacy-compliant way, by assigning greater control to the user. The technique presented in this paper nicely complements current approaches and provides a way to increase the level of privacy protection with an acceptable impact on system performance, offering a solution that certainly supports the real wide-scale deployment of this important novel paradigm.

7. ACKNOWLEDGMENTS

This work was supported in part by the EU, within the 7FP project, under grant agreement 216483 "PrimeLife" and by the Italian MIUR, within PRIN 2006, under project 2006099978 "Basi di dati crittografate". The work of Sushil Jajodia was partially supported by National Science Foundation under grants CT-0716567, CT-0627493, and IIS-0430402 and by Air Force Office of Scientific Research under grant FA9550-07-1-0527.

8. REFERENCES

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: a distributed architecture for secure database services. In *Proc. of CIDR 2005*, Asilomar, CA, January 2005.

[2] R. Agrawal, A. Borgida, and H. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.*, 18(2):253–262, 1989.

[3] R. Agrawal, S. Dar, and H. Jagadish. Direct transitive closure algorithms: design and performance evaluation. *ACM TODS*, 15(3):427–458, 1990.

[4] M. Atallah, K. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *Proc. of the 12th ACM CCS*, Alexandria, VA, November 2005.

[5] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.

[6] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, 2005.

[7] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Proc. of ESORICS 2007*, Dresden, Germany, September 2007.

[8] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. An experimental evaluation of multi-key strategies for data outsourcing. In *Proc. of the 22nd IFIP TC-11 International Information Security Conference*, South Africa, May 2007.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. A data outsourcing architecture combining cryptography and access control. In *Proc. of the 1st Computer Security Architecture Workshop*, Fairfax, VA, November 2007.

[10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: management of access control evolution on outsourced data. In *Proc. of the 33rd VLDB Conference*, Vienna, Austria, September 2007.

[11] K. B. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Trans. Computers*, 55(10):1259–1270, 2006.

[12] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of 18th ICDE*, San Jose, CA, February 2002.

[13] H. Hacigümüs, B. Iyer, and S. Mehrotra. Ensuring integrity of encrypted databases in database as a service model. In *Proc. of the IFIP Conference on Data and Applications Security*, Estes Park Colorado, CA, August 2003.

[14] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the ACM SIGMOD 2002*, Madison, WI, June 2002.

[15] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proc. of the 30th VLDB Conference*, Toronto, Canada, 2004.

[16] K. Irwin and T. Yu. An identifiability-based access control model for privacy protection in open systems. In *Proc. of the WPES 2004*, Washington, DC, October 2004.

[17] H. Jagadish. A compression technique to materialize transitive closure. *ACM TODS*, 15(4):558–598, 1990.

[18] A. Lee, M. Winslett, J. Basney, and V. Welch. The traust authorization service. *ACM TISSEC*, 11(1):1–33, 2008.

[19] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. of the 29th VLDB Conference*, Berlin, Germany, September 2003.

[20] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced database. In *Proc. of the 11th NDSS*, San Diego, CA, February 2004.

[21] M. Narasimha and G. Tsudik. DSAC: integrity for outsourced databases with signature aggregation and chaining. In *Proc. of the 14th ACM ICIKM*, Bremen, Germany, 2005.

[22] R. Sion. Query execution assurance for outsourced databases. In *Proc. of the 31st VLDB Conference*, Trondheim, Norway, September 2005.

[23] H. Wang and L. V. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of the 32nd VLDB Conference*, Seoul, Korea, September 2006.

[24] W. H. Winsborough and N. Li. Safety in automated trust negotiation. *ACM TISSEC*, 9(3):352–390, 2006.

[25] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Using digital credentials on the world wide web. *Journal of Computer Security*, 5(3):255–267, 1997.

[26] M. Winslett, T. Yu, K. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.

[27] T. Yu, X. Ma, and M. Winslett. PRUNES: an efficient and complete strategy for automated trust negotiation over the internet. In *Proc. of the 7th ACM CCS*, Athens, Greece, November 2000.

[28] T. Yu, M. Winslett, and K. Seamons. Interoperable strategies in automated trust negotiation. In *Proc. of the 8th ACM CCS*, Philadelphia, PA, November 2001.

[29] T. Yu, M. Winslett, and K. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM TISSEC*, 6(1):1–42, 2003.