International Journal of Computer Information Systems and Industrial Management Applications.
ISSN2150-7988,Volume 5 (2013) pp. 671-684
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Observation-based Fine Grained Access Control of Data

Raju Halder[1] **and Agostino Cortesi**[2]

[1]Department of Computer Science and Engineering,
Indian Institute of Technology Patna, India
halder@iitp.ac.in

[2]DAIS, Università Ca' Foscari Venezia, Italy
cortesi@unive.it

Abstract: **In this paper, we propose an observation-based fine grained access control (OFGAC) mechanism where data are made accessible at various levels of abstractions according to their sensitivity levels. In this setting, unauthorized users are not able to infer the exact content of the confidential data, while they are allowed to get partial information out of it, according to their access rights. The traditional fine grained access control (FGAC) can be seen as a special case of the OFGAC framework.**

Keywords: Access Control, Abstract Interpretation, Databases

I. Introduction

Fine Grained Access Control (FGAC) mechanism [9, 16, 18, 21, 23, 25, 28, 31, 32] is one of the most effective solutions to ensure the safety of information in databases even at lower levels such as individual tuple/cell level (in case of relational databases) or element/attribute level (in case of XML documents) without changing their original structure. The traditional fine grained disclosure policy p splits any database state into two distinct parts: a public one (insensitive data) and a private one (sensitive data). Generic users are able to see the information from public part only, while the private part remains undisclosed. We denote a database state σ under a disclosure policy p by a tuple $\sigma_p = \langle \sigma_h, \sigma_l \rangle$, where σ_h and σ_l represent the parts of the state corresponding to the private and public part respectively. Given a database state σ_p under a policy p and a query Q, the execution of Q on σ_p returns the result $\xi = S[\![Q]\!](\sigma_p)$. In reality, p depends on the context in which queries are issued, for instance the identity of the issuer, the purpose of the query, the data provider's policy, etc.

As far as the security of the system is concerned, the disclosure policy should comply with the *non-interference* policy [24], *i.e.* the results of admissible queries should not depend on the confidential data in the database. The *non-interference* says that a variation of private (sensitive) database values does not cause any variation of the public (insensitive) view (see Definition 1).

Definition 1 (Non-interference) *Let $\sigma_p = \langle \sigma_h, \sigma_l \rangle$ and $\sigma'_p = \langle \sigma'_h, \sigma'_l \rangle$ be two database states under the disclosure policy p. The non-interference policy says that*

$$\forall Q, \forall \sigma_p, \sigma'_p : \quad \sigma_l = \sigma'_l \implies S[\![Q]\!](\sigma_p) = S[\![Q]\!](\sigma'_p)$$

That is, if the public (insensitive) part of any two database states under disclosure policy p are the same, the execution of any admissible query Q over σ_p and σ'_p return the same results.

In the context of information flow security, the notion of non-interference is too restrictive and impractical in some real systems where intensional leakage of the information to some extent is allowed with the assumption that the power of the external observer is bounded. Thus, we need to weaken or downgrading the sensitivity level of the database information, hence the notion of non-interference which considers weaker attacker model. The weaker attacker model characterizes the observational characteristics of the attacker and can be able to observe specific properties of the private data. Example 1 illustrates this situation with a suitable example.

Example 1 *Consider an XML document that stores customers' information of a bank. Figure 1(a) and 1(b) represent the Document Type Definition (DTD) and its instance respectively. According to the DTD, the document consists of zero or more "customer" elements with three different child elements: "PersInfo", "AccountInfo", "CreditCardInfo" for each customer. The "CreditCardInfo" for a customer is optional, whereas each customer must have at least one bank account represented by "AccountInfo". The element "PersInfo" records the personal information of customers.*

Suppose, according to the access control policy, that employees in bank's customer-care section are not permitted to view the exact content of IBAN and credit-card numbers, while they are allowed to view only the first two digits of IBAN and the last four digits of credit card numbers, keeping other sensitive digits hidden. For instance, in case of the 12 digits credit card number "4023 4581 8419 7835" and the IBAN number "IT10G 02006 02003 000011115996", a customer-

```
<?xml version="1.0"? >
<! DOCTYPE BankCusomers [ >
<! ELEMENT BankCusomers(Customer*) >
<! ELEMENT Customer(PersInfo, AccountInfo+, CreditCardInfo?) >
<! ELEMENT PersInfo(Cid, Name, Address, PhoneNo) >
<! ELEMENT Cid (# PCDATA) >
<! ELEMENT Name (# PCDATA) >
<! ELEMENT Address (street, city, country, pin) >
<! ELEMENT street (# PCDATA) >
<! ELEMENT city (# PCDATA) >
<! ELEMENT country (# PCDATA) >
<! ELEMENT pin (# PCDATA) >
<! ELEMENT PhoneNo (# PCDATA) >
<! ELEMENT AccountInfo (IBAN, type, amount) >
<! ELEMENT IBAN (# PCDATA) >
<! ELEMENT type (# PCDATA) >
<! ELEMENT amount (# PCDATA) >
<! ELEMENT CreditCardInfo (CardNo, ExpiryDate, SecretNo) >
<! ELEMENT CardNo (# PCDATA) >
<! ELEMENT ExpiryDate (# PCDATA) >
<! ELEMENT SecretNo (# PCDATA) >
<! ATTLIST Cid IBAN CDATA #REQUIRED ]>
```

(a) DTD

```
<?xml version="1.0"? >
<BankCusomers>
<Customer>
<PersInfo>
<Cid> 140062 </Cid>
<Name> John Smith </Name>
<Address>
<street> Via Pasini 62 </street>
<city> Venezia </city>
<country> Italy </country>
<pin> 30175 </pin>
</Address>
<PhoneNo> +39 3897745774 </PhoneNo>
</PersInfo>
<AccountInfo>
<IBAN> IT10G 02006 02003 000011115996 </IBAN>
<type> Savings </type>
<amount> 50000 </amount>
</AccountInfo >
<CreditCardInfo>
<CardNo> 4023 4581 8419 7835 </CardNo>
<ExpiryDate> 12/15 </ExpiryDate>
<SecretNo> 165 </SecretNo>
</CreditCardInfo>
</Customer>
</BankCusomers>
```

(b) XML document

Figure. 1: A Document Type Definition (DTD) and its instance

*care personnel is allowed to see them as "**** **** **** 7835" and "IT*** ***** ***** ************" respectively, just to facilitate the searching of credit card number and to redirect the account related issues to the corresponding country (viz, "IT" stands for "Italy"). In addition, suppose the policy specifies that the expiry dates and secret numbers of credit cards and the deposited amounts in accounts are fully-sensitive and completely hidden to them. The traditional FGAC mechanisms are unable to implement this scenario as the IBAN numbers or credit card numbers are neither private nor public as a whole. To implement traditional FGAC, the only possibility is to split the partial sensitive element into two sub-elements: one with private privilege and other with public. For example, the credit-card numbers can be split into two sub-elements: one with first 12 digits which is made private and the other with last 4 digits which is made public. However, practically this is not feasible in all cases, as the sensitivity level and the access-privilege of the same element might be different for different users, and the integrity of data is compromised. For instance, when an integer data (say, 10) is partially viewed as an interval (say, [5, 25]), we can not split it.*

To cope with this situation, we propose an Observation-based Fine Grained Access Control (OFGAC) mechanism where data are made accessible at various levels of abstractions according to their sensitivity levels, based on the Abstract Interpretation framework. In this setting, unauthorized users are not able to infer the exact content of a data cell containing partial sensitive information, while they are allowed to get a relaxed view of it, according to their access rights.

The structure of this paper[1] is as follows: section II recalls some basics on Abstract Interpretation theory and on the concrete and abstract semantics of database query languages. We describe the proposed OFGAC framework to the context of RDBMS and XML documents in sections III and IV respectively. In section V, we discuss the related works in the literature. Finally, in VI, we conclude our work.

[1]The paper is a revised and extended version of [10, 12]

II. Preliminaries

In this section, we recall some basics on the Abstract Interpretation theory [6, 7] and on the concrete and abstract semantics of database query languages [5, 11].

A. Abstract Interpretation Theory

The basic idea of abstract interpretation is that the program behavior at different levels of abstraction is an approximation of its formal concrete semantics. Approximated/abstract semantics is obtained from the concrete one by substituting concrete domains of computation and their basic concrete semantic operations with abstract domains and corresponding abstract semantics operations. The basic intuition is that abstract domains are representations of some properties of interest about concrete domains' values, while abstract operations simulate, over the properties encoded by abstract domains, the behavior of their concrete counterparts.

Abstract interpretation formalizes the correspondence between the concrete semantics $S^c[\![P]\!]$ of syntactically correct program $P \in \mathbb{P}$ in a given programming language $\mathbb{P}$ and its abstract semantics $S^a[\![P]\!]$ which is a safe approximation of the concrete semantics $S^c[\![P]\!]$.

The concrete and abstract semantics domain $\mathfrak{D}^c$ and $\mathfrak{D}^a$ respectively often enjoy stronger properties, such as being complete partial orderings (CPO) or complete lattices.

> *A CPO is a poset $(D, \sqsubseteq)$ where the set D is equipped with an ordering relation $\sqsubseteq$ and satisfies the following property: every sequence of elements*

Table 1: Abstract Syntax of programs embedding SQL statements

	Abstract Syntax
Syntactic Sets	$c ::= n \mid k$
$n : \mathbb{Z}$ (Integer)	$e ::= c \mid v_d \mid v_a \mid op_u\ e \mid e_1\ op_b\ e_2$, where $op_u \in \{+,-\}$ and $op_b \in \{+,-,*,/,\ldots\ldots\}$
$k : \mathbb{S}$ (String)	$b ::= e_1\ op_r\ e_2 \mid \neg b \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid true \mid false$, where $op_r \in \{=,\geq,\leq,<,>,\neq,\ldots\}$
$c : \mathbb{C}$ (Constants)	$\tau ::= c \mid v_a \mid v_d \mid f_n(\tau_1,\tau_2,...,\tau_n)$, where f_n is an n-ary function.
$v_a : \mathbb{V}_a$ (Application Variables)	$a_f ::= R_n(\tau_1,\tau_2,...,\tau_n) \mid \tau_1 = \tau_2$, where $R_n(\tau_1,\tau_2,...,\tau_n) \in \{true, false\}$
$v_d : \mathbb{V}_d$ (Database Variables)	$\phi ::= a_f \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \forall x_i\ \phi \mid \exists x_i\ \phi$
$v : \mathbb{V} \triangleq \mathbb{V}_d \cup \mathbb{V}_a$ (Variables)	$g(\vec{e}) ::=$ `GROUP BY`$(\vec{e}) \mid id$
$e : \mathbb{E}$ (Arithmetic Expressions)	$r ::=$ `DISTINCT` $\mid$ `ALL`
$b : \mathbb{B}$ (Boolean Expressions)	$s ::=$ `AVG` $\mid$ `SUM` $\mid$ `MAX` $\mid$ `MIN` $\mid$ `COUNT`
$A : \mathbb{A}$ (Action)	$h(e) ::= s \circ r(e) \mid$ `DISTINCT`$(e) \mid id$
$\tau : \mathbb{T}$ (Terms)	$h(*) ::=$ `COUNT(*)`
$a_f : \mathbb{A}_f$ (Atomic Formulas)	$\vec{h}(\vec{x}) ::= \langle h_1(x_1),...,h_n(x_n)\rangle$, where $\vec{h} = \langle h_1,...,h_n\rangle$ and $\vec{x} = \langle x_1,...,x_n\rangle$
$\phi : \mathbb{W}$ (Pre-condition)	$f(\vec{e}) ::=$ `ORDER BY ASC`$(\vec{e}) \mid$ `ORDER BY DESC`$(\vec{e}) \mid id$
$Q : \mathbb{Q}$ (SQL statements)	$A ::= select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi,\ g(\vec{e})) \mid update(\vec{v_d},\ \vec{e}) \mid insert(\vec{v_d},\ \vec{e}) \mid delete(\vec{v_d})$
$I : \mathbb{I}$ (Program statements)	$Q ::= \langle A,\phi\rangle \mid Q'$ `UNION` $Q'' \mid Q'$ `INTERSECT` $Q'' \mid Q'$ `MINUS` Q''
	$I ::= skip \mid v_a := e \mid v_a :=? \mid Q \mid if\ b\ then\ I_1\ else\ I_2 \mid while\ b\ do\ I \mid I_1; I_2$

$x_0 \sqsubseteq x_1 \sqsubseteq ... \sqsubseteq x_n$ in D has a limit or least upper bound in D, that is, there is an element in $x \in D$ (written as $\bigsqcup_i x_i$) such that, (i) $\forall x_i,\ x_i \sqsubseteq x$, (ii) if x' is any other upper bound for the x_i, then $x \sqsubseteq x'$.

A poset $(D, \sqsubseteq)$ is called a complete lattice (denoted $\langle D, \sqsubseteq, \sqcup, \sqcap, \top, \bot\rangle$) if every subset S of D has a least upper bound (written $\sqcup S$) and a greatest lower bound (written $\sqcap S$) in it, and it is bounded by a unique largest element $\top = \sqcup D$ and a unique smallest element $\bot = \sqcap D$.

The correspondence between the concrete and abstract semantics domains $\mathfrak{D}^c$ and $\mathfrak{D}^a$ is provided by a Galois Connection $(\mathfrak{D}^c, \alpha, \gamma, \mathfrak{D}^a)$, where the function $\alpha : \mathfrak{D}^c \longrightarrow \mathfrak{D}^a$ and $\gamma : \mathfrak{D}^a \longrightarrow \mathfrak{D}^c$ form an adjunction, *i.e.* $\forall A \in \mathfrak{D}^a, \forall C \in \mathfrak{D}^c : \alpha(C) \sqsubseteq^a A \Leftrightarrow C \sqsubseteq^c \gamma(A)$. The functions α and γ are called abstraction and concretization maps respectively.
Let (C, α, γ, A) be a Galois connection, $f : C \rightarrow C$ be a concrete function and $f^\sharp : A \rightarrow A$ be an abstract function. $f^\sharp$ is a sound, i.e., correct approximation of f if $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$. When the soundness condition is strengthened to equality, i.e., when $f \circ \gamma = \gamma \circ f^\sharp$, the abstract function $f^\sharp$ is a complete approximation of f in A. This means that no loss of precision is accumulated in the abstract computation through $f^\sharp$.

B. Semantics of Query Languages

An application embedding SQL statements basically interacts with two worlds: *user world* and *database world*. Corresponding to these two worlds there exist two sets of variables: database variables $\mathbb{V}_d$ and application variables $\mathbb{V}_a$. Variables from $\mathbb{V}_d$ are involved only in the SQL statements, whereas variables in $\mathbb{V}_a$ may occur in all types of instructions of the application. Any SQL statement Q is denoted by a tuple $Q = \langle A, \phi\rangle$ where A and ϕ refer to *action part* and *pre-condition part* of Q respectively. A SQL statement Q first identifies an active data set from the database using the pre-condition ϕ, and then performs the appropriate operations on that data set using the SQL action A. The pre-condition ϕ appears in SQL statements as a well-formed formula in first-order logic. Table 1 depicts the syntactic sets and the abstract syntax of programs embedding SQL statements.

Program Environments. The SQL embedded program P acts on a set of constants $const(P) \in \wp(\mathbb{C})$ and set of variables $var(P) \in \wp(\mathbb{V})$, where $\mathbb{V} \triangleq \mathbb{V}_d \cup \mathbb{V}_a$. These variables take their values from semantic domain $\mathfrak{D}_\mho$, where $\mathfrak{D}_\mho = \{\mathfrak{D} \cup \{\mho\}\}$ and $\mho$ represents the undefined value.
Now we define two environments $\mathfrak{E}_d$ and $\mathfrak{E}_a$ corresponding to the database and application variable sets $\mathbb{V}_d$ and $\mathbb{V}_a$ respectively.
Definition 2 (Application Environment) *An application environment $\rho_a \in \mathfrak{E}_a$ maps a variable $v \in dom(\rho_a) \subseteq \mathbb{V}_a$ to its value $\rho_a(v)$. So, $\mathfrak{E}_a \triangleq \mathbb{V}_a \longmapsto \mathfrak{D}_\mho$.*
Definition 3 (Database Environment) *We consider a database as a set of indexed tables $\{t_i \mid i \in I_x\}$ for a given set of indexes I_x. We define database environment by a function ρ_d whose domain is I_x, such that for $i \in I_x$, $\rho_d(i) = t_i$.*
Definition 4 (Table Environment) *Given a database environment ρ_d and a table $t \in d$. We define $attr(t) = \{a_1, a_2, ..., a_k\}$. So, $t \subseteq D_1 \times D_2 \times .... \times D_k$ where, a_i is the attribute corresponding to the typed domain D_i. A table environment ρ_t for a table t is defined as a function such that for any attribute $a_i \in attr(t)$,*

$$\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t\rangle$$

Where π is the projection operator, i.e. $\pi_i(l_j)$ is the i^{th} element of the l_j-th row. In other words, ρ_t maps a_i to the ordered set of values over the rows of the table t.

Small-Steps Operational Semantics. Let Σ be a set of states defined by $\Sigma \triangleq \mathfrak{E}_d \times \mathfrak{E}_a$, where $\mathfrak{E}_d$ and $\mathfrak{E}_a$ denote the set of all database environments and the set of all application environments respectively. That is, a state $\sigma \in \Sigma$ is denoted by a tuple $\langle \rho_d, \rho_a\rangle$ where $\rho_d \in \mathfrak{E}_d$ and $\rho_a \in \mathfrak{E}_a$ are the database environment and application environment respectively. The set of states of P is, thus, defined as:

$$\Sigma[\![P]\!] \triangleq \mathfrak{E}_d[\![P]\!] \times \mathfrak{E}_a[\![P]\!]$$

where $\mathfrak{E}_d[\![P]\!]$ and $\mathfrak{E}_a[\![P]\!]$ are the set of database and application environments of the program P whose domain is the set of program variables.
The transition relation $S \in (\mathbb{I} \times \Sigma) \mapsto \wp(\Sigma)$ specifies which successor states $\langle \rho_d', \rho_a'\rangle \in \Sigma$ can follow when a

statement $I \in \mathbb{I}$ executes on state $\langle \rho_d, \rho_a \rangle \in \Sigma$. Therefore, the transitional semantics $S[\![P]\!] \in (P \times \Sigma[\![P]\!]) \mapsto \wp(\Sigma[\![P]\!])$ of a program $P \in \mathbb{P}$ restricts the transition relation to program instructions, *i.e.*

$$S[\![P]\!](\langle \rho_d, \rho_a \rangle) = \{\langle \rho'_d, \rho'_a \rangle \mid \langle \rho_d, \rho_a \rangle, \langle \rho'_d, \rho'_a \rangle \in \Sigma[\![P]\!] \wedge \langle \rho'_d, \rho'_a \rangle \in S[\![I]\!](\rho_d, \rho_a) \wedge I \in P\}$$

Lifting the semantics to abstract domains. We lift the concrete semantics of programs embedding SQL statements to an abstract setting by introducing the notion of abstract databases, where instead of working on the concrete databases, abstract versions of the queries are applied to abstract databases in which some information are disregarded and concrete values are possibly represented by suitable abstractions.

Definition 5 (Abstract Databases) *Let dB be a database. The database $dB^\sharp = \alpha(dB)$ where α is the abstraction function, is said to be an abstract version of dB if there exist a representation function γ, called concretization function such that for all tuple $\langle x_1, x_2, \ldots, x_n \rangle \in dB$ there exist a tuple $\langle y^\sharp_1, y^\sharp_2, \ldots, y^\sharp_n \rangle \in dB^\sharp$ such that $\forall i \in [1 \ldots n]$ $(x_i \in id(y^\sharp_i) \vee x_i \in \gamma(y^\sharp_i))$, where id represents identity function.*

The abstract version corresponding to all concrete functions such as `Group By`, `Order By`, Aggregate Functions, Set Operations, etc are defined in such a way so as to preserve the soundness condition. For more details, see [11].

Given an abstraction, let T and $T^\sharp$ be a concrete and abstract table respectively. The correspondence between T and $T^\sharp$ are described using the concretization and abstraction maps γ and α respectively. If Q_{sql} and $Q^\sharp$ are representing the SQL queries on concrete and abstract domain respectively, let T_{res} and $T^\sharp_{res}$ are the results of applying Q_{sql} and $Q^\sharp$ on the T and $T^\sharp$ respectively. The following fact illustrate the soundness condition of abstraction:

$$\begin{tikzcd}
T \arrow[r, "Q_{sql}"] & T_{res} \sqsubseteq \gamma(T^\sharp_{res}) \\
T^\sharp \arrow[u, "\gamma"'] \arrow[r, "Q^\sharp"'] & T^\sharp_{res} \arrow[u, "\gamma"']
\end{tikzcd}$$

Lemma 1 (Soundness of SQL [11]) *Let $T^\sharp$ be an abstract table and $Q^\sharp$ be an abstract query. $Q^\sharp$ is sound if $\forall T \in \gamma(T^\sharp)$. $\forall Q_{sql} \in \gamma(Q^\sharp) : Q_{sql}(T) \subseteq \gamma(Q^\sharp(T^\sharp))$.*

III. OFGAC for RDBMS

In this section, we describe the OFGAC framework for RDBMS, in order to provide various levels of accessibility to database information.

A. Observation-based Access Control Policy Specification

Let us define observation-based access control policy for RDBMS under OFGAC framework, in contrast to traditional binary-based access control policy.

Table 2: Concrete Database *dB*

(a) *"emp"*

eID	*Name*	*Age*	*Dno*	*Sal*
1	Matteo (N)	30	2	2800 (N)
2	Pallab (N)	22	1	1500
3	Sarbani (N)	56 (N)	2	2300
4	Luca (N)	35	1	6700 (N)
5	Tanushree (N)	40 (N)	3	4900
6	Andrea (N)	52 (N)	1	7000 (N)
7	Alberto (N)	48	3	800
8	Mita (N)	29 (N)	2	4700 (N)

(b) *"dept"*

Dno	*Name*	*Loc*	*Phone*	*DmngrID*
1	Financial	Venice	111-1111 (N)	6
2	Research	Rome	222-2222 (N)	8
3	Admin	Treviso	333-3333 (N)	3

Table 3: Partial Abstract Database $dB^\sharp$

(a) *"emp$^\sharp$"*

eID$^\sharp$	*Name$^\sharp$*	*Age$^\sharp$*	*Dno$^\sharp$*	*Sal$^\sharp$*
1	Male	30	2	Medium
2	Male	22	1	1500
3	Female	[50, 59]	2	2300
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
7	Male	48	3	800
8	Female	[20, 29]	2	High

(b) *"dept$^\sharp$"*

Dno$^\sharp$	*Name$^\sharp$*	*Loc$^\sharp$*	*Phone$^\sharp$*	*DmngrID$^\sharp$*
1	Financial	Venice	$\top$	6
2	Research	Rome	$\top$	8
3	Admin	Treviso	$\top$	3

Definition 6 (Observation-based Disclosure Policy) *Given a domain of observable properties D, and an abstraction function $\alpha_D : \wp(val) \to D$, an observation-based disclosure policy op assigned to the observer O is a tagging that assigns each value v in the database state σ a tag $\alpha_D(v) \in D$, meaning that O is allowed to access the value $\alpha_D(v)$ instead of its actual value v.*

Unlike traditional FGAC, the database information which is unauthorized under an observation-based disclosure policy *op*, is abstracted by the information at various levels of abstractions representing some properties of interest, rather than NULL or special symbols [20, 28]. The levels of abstractions depend on the sensitivity level of information: less sensitive values are abstracted by lower level of abstraction, while more sensitive values are abstracted by higher level of abstraction. The unauthorized users, therefore, could not be able to infer the exact content of sensitive cells. This way, we can tune different parts of same database to different levels of abstractions at the same time, giving rise to various observational access control for various parts. The query issued by the external users will be directed to and executed over the abstract databases, yielding to a sound approximation of the query results.

Example 2 *Consider the database dB of Table 2 where cells containing sensitive information are marked by '(N)'. Under observation-based disclosure policy, we abstract these sensi-*

tive information by abstract values resulting into an abstract database $dB^\sharp$ depicted in Table 3. Observe that in $emp^\sharp$ the ages are abstracted by the elements from the domain of intervals, the salaries are abstracted by their relative measures: Low, Medium, High, Very High, and the names are abstracted by their sex properties. It is worthwhile to mention here that we assume salaries to be more sensitive than ages, and so we abstract salary values with higher level of abstraction, although both are numeric. Since phone numbers of all departments are strictly confidential, they are abstracted by the top element $\top$ of their corresponding abstract lattice depicted in $dept^\sharp$. We call the resulting database as "Partial Abstract Database", in contrast to "Full Abstract Database", since only a subset of the database information is abstracted. The correspondence between the concrete and abstract values of salaries can formally be expressed by the abstraction and concretization functions α_{sal} and γ_{sal} respectively as follows:

$$\alpha_{sal}(X) \triangleq \begin{cases} \bot \ \ if\, X = \emptyset \\ Low \ \ if\, \forall x \in X: \ 500 \le x \le 1999 \\ Medium \ \ if\, \forall x \in X: \ 2000 \le x \le 3999 \\ High \ \ if\, \forall x \in X: \ 4000 \le x \le 5999 \\ Very\ High \ \ if\, \forall x \in X: \ 6000 \le x \le 10000 \\ \top \ otherwise \end{cases}$$

$$\gamma_{sal}(d) \triangleq \begin{cases} \emptyset \ if\ d = \bot \\ \{x \ : \ 500 \le x \le 1999\} \ if\, d = Low \\ \{x \ : \ 2000 \le x \le 3999\} \ if\, d = Medium \\ \{x \ : \ 4000 \le x \le 5999\} \ if\, d = High \\ \{x \ : \ 6000 \le x \le 10000\} \ if\, d = Very\ High \\ \{x \ : \ 500 \le x \le 10000\} \ if\, d = \top \end{cases}$$

Formally, the sensitive values of a data cell belonging to an attribute x are abstracted by using the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \to D_x^{abs}$ and $\gamma_x : D_x^{abs} \to \wp(D_x^{con})$) respectively. In case of insensitive information, the abstraction and concretization functions represent the identity function *id*.

Given a concrete database state σ_{op} under an observation-based disclosure policy *op*, the abstract state is obtained by performing $\sigma_{op}^\sharp = \alpha(\sigma_{op})$ where the abstraction function α can be expressed as collection of abstraction functions for all attributes in the database.

We assume that for each type of values in a database there exists a hierarchy of abstractions such that Galois Connections combine consistently, *i.e.* if $(X, \alpha_1, \gamma_1, Y)$ and $(Y, \alpha_2, \gamma_2, Z)$ represent two Galois Connection, then we have the following:

$$if\ (X, \alpha_1, \gamma_1, Y)\ and\ (Y, \alpha_2, \gamma_2, Z)\ then\ (X, \alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2, Z)$$

Observe that the traditional FGAC [20, 28] is a special case of our OFGAC framework where each unauthorized cell is abstracted by the top element $\top$ of its corresponding abstract lattice.

Table 4: Preserving Referential Integrity Constraint by using Type-2 variable

(a) *"Supplier"*

S-id	*Name*	*Age*
S230 (N)	Alice	24
S201 (N)	Bob	21
S368 (N)	Tea	22

(b) *"Supp – Part"*

S-id	*P-id*	*QTY*
S230 (N)	P140 (N)	120
S201 (N)	P329 (N)	260 (N)
S230 (N)	P563 (N)	200
S368 (N)	P329 (N)	450 (N)
S368 (N)	P140 (N)	430 (N)

(c) *"Part"*

P-id	*Pname*
P140 (N)	Screw
P329 (N)	Bolt
P563 (N)	Nut

(d) *"Supplier$^\sharp$"*

S-id$^\sharp$	*Name$^\sharp$*	*Age$^\sharp$*
$(v_1, [S200, S249])$	Alice	24
$(v_2, [S200, S249])$	Bob	21
$(v_3, [S350, S399])$	Tea	22

(e) *"Supp – Part$^\sharp$"*

S-id$^\sharp$	*P-id$^\sharp$*	*QTY$^\sharp$*
$(v_1, [S200, S249])$	$(v_4, [P100, P149])$	120
$(v_2, [S200, S249])$	$(v_5, [P300, P349])$	[250,299]
$(v_1, [S200, S249])$	$(v_6, [P550, P599])$	200
$(v_3, [S350, S399])$	$(v_5, [P300, P349])$	[450, 499]
$(v_3, [S350, S399])$	$(v_4, [P100, P149])$	[400, 449]

(f) *"Part$^\sharp$"*

P-id$^\sharp$	*Pname$^\sharp$*
$(v_4, [P100, P149])$	Screw
$(v_5, [P300, P349])$	Bolt
$(v_6, [P550, P599])$	Nut

B. Preserving Referential Integrity Constraints

In OFGAC framework, the uniqueness criterion for primary and foreign key attributes could not be maintained due to the loss of precision of data values while abstracting, and therefore the referential integrity constraints among database relations might be hampered. In [28], Wang et al. used Type-2 variable in order to keep these referential integrity constraints intact while masking operation is performed.

We extend the same approach of Wang et al. [28] in our OFGAC framework. We denote the Type-2 variable by a pair (v, A) where A is an abstract value and $\gamma(A)$ is the domain of the variable v, depicted in definition 7.

Definition 7 (Type-2 Variable) *A Type-2 variable is represented by a pair $(v,\ A)$ where A is an abstract value and $\gamma(A)$ is the domain of the variable v. Given v_1, v_2, A_1, A_2 where v_1 is assumed to be different from v_2, then we have that (i) "$(v_1, A_1) = (v_1, A_1)$" and "$(v_1, A_1) \neq (v_2, A_1)$" are always true, (ii) "$(v_1, A_1) \neq (v_2, A_2)$" is true if $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, (iii) "$(v_1, A_1) = (v_2, A_2)$" is $\top$ if $\gamma(A_1) \cap \gamma(A_2) \neq \emptyset$, and (iv) "$(v_1, A_1)$=$c$" is $\top$ where $c \in \gamma(A_1)$.*

Given two sensitive values x_1 and x_2 under the same primary/foreign key attribute. According to Definition 7, we abstract them as follows: (*i*) if $x_1 = x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then both x_1 and x_2 are abstracted by the Type-2 variable (v, A), (*ii*) if $x_1 \neq x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then x_1 and x_2 are abstracted by (v_1, A) and (v_2, A) respectively, (*iii*) if $x_1 \neq x_2$, $\alpha(x_1) = A_1$, $\alpha(x_2) = A_2$ and $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, then x_1 and x_2 are abstracted by (v_1, A_1) and (v_2, A_2) respectively.

Example 3 *Consider the supplier-parts database and its abstract version under an observation-based access control policy, depicted in Table 4. The attributes S-id and P-id are*

Table 5: $\xi_1^\sharp$: Result of $Q_1^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Sal*$^\sharp$
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

the primary keys of the tables "Supplier" and "Part" respectively, whereas the composite attribute {S-id, P-id} is used as the primary key of the table "Supp-Part". Observe that S-id and P-id in "Supp-Part" are used as the foreign keys that link to the primary keys of "Supplier" and "Part" respectively, and relate the suppliers with the parts sold by them. Suppose according to the disclosure policy that all values of the attributes S-id, P-id and some values of QTY in "Supp-Part" are confidential (marked with 'N' in the parenthesis). If we abstract these values by the abstract values from the domain of intervals, we may loose the ability to identify the tuples uniquely and the secure linking between "Supplier" and "Part" might be disturbed. To preserve the uniqueness of the values in abstract domain, we use Type-2 variable, as depicted in the abstract tables "Supplier$^\sharp$*", "Part*$^\sharp$*" and "Supp-Part*$^\sharp$*". Observe that since the attribute QTY is not primary key or foreign key, we abstract them only by the abstract values from the domain of intervals.*

C. Query Evaluation under OFGAC

A general framework for Abstract Interpretation of Relational Databases has been introduced in [11]. Here, we briefly recall some notions on query abstraction, and we extend them by considering queries on multiple abstractions as well.

Example 4 *Consider the concrete database of Table 2 and the corresponding partial abstract database depicted in Table 3 under an observational disclosure policy op. Suppose an external user issues a query Q_1 under op as below:*

Q_1 =`SELECT * FROM emp WHERE Sal>4800;`

The system transforms Q_1 into the corresponding abstract version of the query (denoted $Q_1^\sharp$) as shown below:

$Q_1^\sharp$ =`SELECT * FROM emp`$^\sharp$ `WHERE Sal`$^\sharp$ `> 4800 OR Sal`$^\sharp$ `>`$^\sharp$ `High;`

The result of $Q_1^\sharp$ on emp$^\sharp$ is depicted in Table 5. Observe that the pre-condition $\phi^\sharp$ (represented by WHERE clause in $Q_1^\sharp$) evaluates to true for the first three tuples in the result $\xi_1^\sharp$, whereas it evaluates to $\top$ (may be true or may be false) for the last tuple. The result of $Q_1^\sharp$ is sound as it over-approximate the result of the query Q_1. Observe in fact that $Q_1^\sharp$ includes also the "false positive" corresponding to the concrete information about Mita.

In [11], we defined the abstract aggregate functions and abstract set operations in an abstract domain of interest. In OFGAC framework, we apply them in the same way as depicted in Example 5 and 6 respectively.

Example 5 *Consider the abstract database of Table 3 and an abstract query $Q_2^\sharp$ containing aggregate functions.*

$Q_2^\sharp$ =`SELECT COUNT`$^\sharp$`(*), AVG`$^\sharp$`(Age`$^\sharp$`) FROM emp`$^\sharp$ `WHERE (Age`$^\sharp$ `BETWEEN 32 AND 55) OR (Age`$^\sharp$ `BETWEEN`$^\sharp$ `[30, 39] AND [50, 59]);`

The result of $Q_2^\sharp$ on emp$^\sharp$ is depicted in Table 6. In the example, the evaluation of the abstract WHERE clause extracts five tuples in total where three tuples with eID$^\sharp$ equal to 4, 5, 7 belong to $G_{yes}^\sharp$ and two tuples with eID$^\sharp$ equal to 3, 6 belong to $G_{may}^\sharp$. Thus, in case of **AVG**$^\sharp$*(Age$^\sharp$), we get $a^\sharp$ = $fn^\sharp$({35, [40, 49], 48}) = average$^\sharp$({35, [40, 49], 48})=[41, 44] and $b^\sharp$ = $fn^\sharp$({[50, 59], 35, [40, 49], [50, 59], 48}) = average$^\sharp$({[50, 59], 35, [40, 49], [50, 59], 48}) = [44, 50]. Hence,* **AVG**$^\sharp$*(Age$^\sharp$) = [min$^\sharp$($a^\sharp$), max$^\sharp$($b^\sharp$)]=[41, 50]. Similarly, in case of* **COUNT**$^\sharp$*(∗), we get $a^\sharp$ = count$^\sharp$($G_{yes}^\sharp$)=[3, 3] and $b^\sharp$ = count$^\sharp$($G^\sharp$) = [5, 5]. Thus,* **COUNT**$^\sharp$*(∗) =[3, 5]. Observe that the result is sound, i.e., $\xi_2 \in \gamma(\xi_2^\sharp)$ where ξ_2 is the result of a concrete query $Q_2 \in \gamma(Q_2^\sharp)$.*

Table 6: $\xi_2^\sharp$: Result of $Q_2^\sharp$

COUNT$^\sharp$(*)	AVG$^\sharp$(*Age*$^\sharp$)
[3, 5]	[41, 50]

Table 7: Abstract Computation of $Q_3^\sharp$

(a) $\xi_l^\sharp$: Result of $Q_l^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Sal*$^\sharp$
1	Male	30	2	Medium
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(b) $\xi_r^\sharp$: Result of $Q_r^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Sal*$^\sharp$
4	Male	35	1	Very high
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(c) $\xi_3^\sharp$: Performing MINUS$^\sharp$ between $\xi_l^\sharp$ & $\xi_r^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Sal*$^\sharp$
1	Male	30	2	Medium
5	Female	[40, 49]	3	4900
8	Female	[20, 29]	2	High

Example 6 *Consider the abstract database of Table 3 and an abstract query $Q_3^\sharp$ = $Q_l^\sharp$* **MINUS**$^\sharp$ *$Q_r^\sharp$, where*

$Q_l^\sharp$ =`SELECT * FROM emp`$^\sharp$ `WHERE Sal`$^\sharp$ `> 2500 OR Sal`$^\sharp$ `>`$^\sharp$ `Medium;`

$Q_r^\sharp$ =`SELECT * FROM emp`$^\sharp$ `WHERE Sal`$^\sharp$ `> 5500 OR Sal`$^\sharp$ `>`$^\sharp$ `High;`

The execution of $Q_l^\sharp$ and $Q_r^\sharp$ on emp$^\sharp$ are depicted in Table 7(a) and 7(b) respectively. In Table 7(a), for the first tuple the pre-condition of $Q_l^\sharp$ evaluates to $\top$ (thus, belongs to $\xi_{may_l}^\sharp$), whereas for the remaining four tuples it evaluates to true (thus, belongs to $\xi_{yes_l}^\sharp$). Similarly, in Table 7(b), for the first two tuples the pre-condition of $Q_r^\sharp$ evaluates to true (hence, belongs to $\xi_{yes_r}^\sharp$), whereas for the last one it evaluates to $\top$ (hence, belongs to $\xi_{may_r}^\sharp$). Thus, the first component

$(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r})) \in \xi^\sharp_3$ contains the tuple with eID$^\sharp$ equal to 5, and the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r}) \in \xi^\sharp_3$ contains the tuples with eID$^\sharp$ equal to 1 and 8, as shown in Table 7(c). The result is sound, i.e., $\xi_3 \in \gamma(\xi^\sharp_3)$ where ξ_3 is the result of a concrete query $Q_3 \in \gamma(Q^\sharp_3)$.

D. Collusion Attacks

Wang et al. in [28] illustrate the security of the FGAC system in case of collusion and multi-query attacks. They define the security aspect in the context of *one-party single-query/weak security* and *multi-party multi-query/strong security*, and prove that the system with weak-security is also secure under strong-security.

In OFGAC, transforming the system into an abstract domain means transforming the attackers, and the attackers are modeled by abstractions. The robustness of the database under OFGAC policies depends on the ability of the external observers to distinguish the database states based on the observable properties of the query results.

Here we consider three different scenarios: Figure 2(a), 2(b) and 2(c) illustrates these three cases where the shaded portions indicate the sensitive information and α ($\alpha_i \neq \alpha_j$ if $i \neq j$) is the abstraction function used to abstract those sensitive information.

Case 1: Multiple Policies/Single Abstraction:

Suppose each of the n observers under observation-based policies $op_1, op_2, \ldots, op_n$ respectively issue a query Q. Given a database state σ without any policy, we denote by $\sigma^\sharp_{op_i}$ an abstract database state under op_i (where $i = 1, \ldots, n$). Therefore, observer O_i under op_i will get the query result $\xi^\sharp_i = S^\sharp[\![Q^\sharp]\!](\sigma^\sharp_{op_i})$ where $Q^\sharp$ is the abstract version of Q. When these n users collude, they feed the query results $\xi^\sharp_i$, $i = \ldots, n$, to a function f which can perform some comparison or computation (*viz*, difference operation) among the results and infer about some sensitive information for some observers.

For instance, suppose a portion of database information is sensitive under policy op_j, while it is insensitive under another policy op_k, $j \neq k$. In the former case, this part of information will be abstracted, while in the latter case it will not. Thus, if this portion of information appears in both of the query results $\xi^\sharp_j$ and $\xi^\sharp_k$, then it is possible for the j^{th} observer to infer the exact content of that portion of information as it is not abstracted in ξ_k.

Let $\sigma_{op} = \langle \sigma_l, \sigma_h \rangle$ and $\sigma_{op'} = \langle \sigma'_l, \sigma'_h \rangle$ be the database states under two different policies op and op'. The database state $\sigma_{op \bullet op'}$ obtained by combining two policies op and op' are defined as follows:

$$\sigma_{op \bullet op'} = \langle ((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h)), (\sigma_h \cap \sigma'_h) \rangle$$

This fact is depicted in Figure 3. So, when the observers under op and op' collude and share the

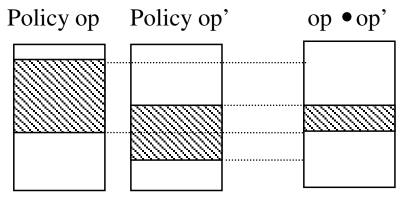


Figure. 3: Combination of policies

query results, both will act as equivalent to the observer under the policy ($op \bullet op'$) and thus they can infer the values belonging to the public part of $op \bullet op'$, *i.e.*, $((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h))$ by issuing a sequence of queries individually and by comparing the results together.

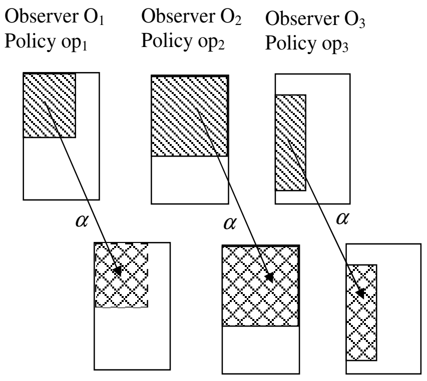
Case 2: Single Policy/Multiple Abstraction:

Consider n different observers O_1, O_2, $\ldots$, O_n under the same policy op and the sensitive information part is abstracted to different level of abstraction to different observers. Higher levels of abstraction make the database information less precise, whereas lower levels of abstraction represent them with more precision. Thus, the result of a query for the one with higher abstraction contains less precise information than that with lower abstraction.

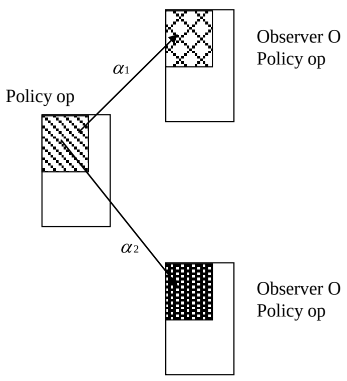
Consider two different observers O_1 and O_2 under op where the sensitive database information of σ_{op} are abstracted by the domains of abstraction D^{abs}_1 and D^{abs}_2, yielding to $\sigma^{1\sharp}_{op}$ and $\sigma^{2\sharp}_{op}$ respectively.

First consider the case where D^{abs}_2 is a more abstract domain than D^{abs}_1, *i.e.*, D^{abs}_2 is an abstraction of D^{abs}_1. Since both observers are under the same policy, the query results over $\sigma^{1\sharp}_{op}$ and $\sigma^{2\sharp}_{op}$ may contain some common abstract information - one from D^{abs}_1 and other from D^{abs}_2. Thus when O_1 and O_2 collude, it is possible for O_2 to obtain sensitive information with lower level of abstraction from the result obtained by O_1 as it is abstracted with lower level of abstraction for O_1. But no real collusion may arise in this case, as the overall information available to O_1 and O_2 together is at most as precise as the one already available to O_1.

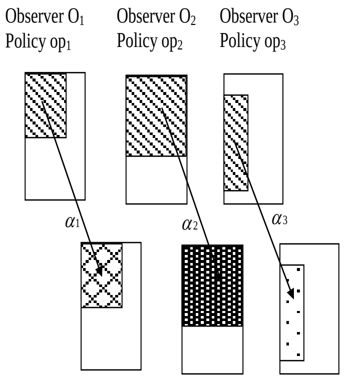
The other case is where the two domains are not one the abstraction of the other. For example, let in a particular database state an attribute of a table has the sensitive values represented by an ordered list ⟨5, 0, 2, 3, 1⟩. Suppose the observer O_1 is limited by the property DOM represented by domain of intervals as shown in Figure 4(a), while O_2 is limited by parity property represented by the abstract domain PAR={⊥, EVEN, ODD, ⊤} as depicted in Figure 4(b). Thus O_1 sees ⟨[4, 5], [0, 1], [2, 3], [2, 3], [0, 1]⟩, while O_2 sees ⟨ODD, EVEN, EVEN, ODD, ODD⟩. When O_1 and O_2 collude, they can infer the exact values for the attribute, *i.e.*, ⟨5, 0, 2, 3, 1⟩ by combining the query results. The corresponding



(a) Multiple policies/Single abstraction



(b) Single policy/Multiple abstraction



(c) Multiple policies/Multiple abstraction

Figure. 2: Policies and Observations

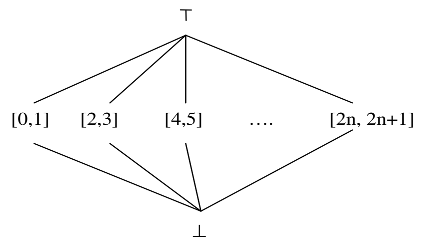
combined lattice obtained by reduced product [7] of the above two abstract lattices DOM and PAR is shown in Figure 5(a).

Given an OFGAC under *Single policy/Multiple level abstraction* scenario where same information under the same policy is abstracted by n different level of abstraction to n different observers. Such OFGAC is *collusion-prone* when intersection of the sets (not singletons) obtained by concretizing abstract values of any common sensitive cell appearing in different query results for different observers, yield to a singleton. This is depicted in Definition 8.

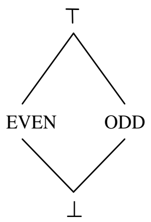
We now show an example where no collusion takes place in practice. Consider two observers O_1 and O_2, where O_1 is limited by the sign property depicted in Figure 4(c), whereas O_2 is limited by the parity property depicted in Figure 4(b). Let ⟨-2, 0, 2, -1, 1⟩ be a list of sensitive values appearing in the results of the queries issued by both O_1 and O_2. Thus, O_1 sees ⟨-, +, +, -, +⟩, while O_2 sees ⟨EVEN, EVEN, EVEN, ODD, ODD⟩. When O_1 and O_2 collude, they can infer the values as ⟨EVEN^-, EVEN^+, EVEN^+, ODD^-, ODD^+⟩ by combining the query results. However, although these combined abstract values represent more precise information than that of the individual result, the observer still could not be able to infer the exact content. Figure 5(b) shows the combined abstract lattice obtained by reduced product [7] of two abstract lattices SIGN and PAR.

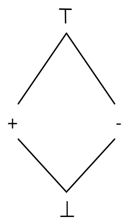
Definition 8 *An OFGAC under Single policy/Multiple level abstraction scenario is collusion-prone, if the OFGAC uses n different abstract domains* $D_1^{abs}, \ldots, D_n^{abs}$ *for n different observers and* $\exists\{d_i, \ldots, d_j\} \in D_i^{abs} \times \cdots \times D_j^{abs}$ *for* $\{i, \ldots, j\} \subseteq \{1, \ldots, n\}$ *such that* $\bigcap_{k \in \{i,\ldots,j\}} \gamma(d_k) = \{e\}$ *while* $\forall k \in \{i, \ldots, j\}$, $\gamma(d_k) \neq \{e\}$.

Theorem 1 *An OFGAC using n different abstract domains* $D_1^{abs}, \ldots, D_n^{abs}$ *for n different observers under the same policy is collusion-prone if the reduced product [7] of* $\{D_i^{abs}, \ldots, D_j^{abs}\} \subseteq \{D_1^{abs}, \ldots, D_n^{abs}\}$ *is isomorphic to a concrete domain D.*



(a) DOM





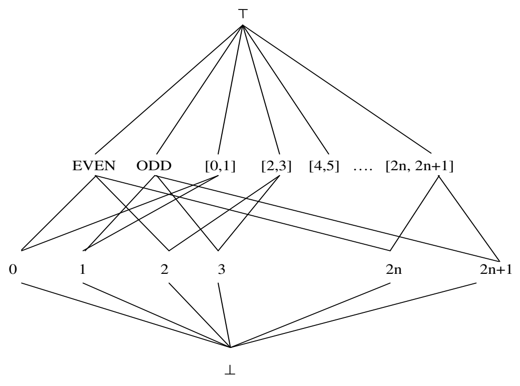
(b) PAR (c) SIGN

Figure. 4: Abstract Lattices of DOM, PAR and SIGN

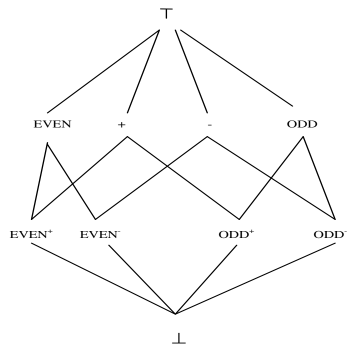
Case 3: Multiple Policies/Abstractions: This is the combination of the previous two cases. Observers may collude to act as the observer under the combination of their individual policies, or may try to infer about the confidential information appearing in the query results by combining (*e.g.*, intersecting) their domain of abstract values.

IV. OFGAC for XML

The notion of OFGAC can be extended to the context of XML documents. In order to do that, we first introduce the notion of access control policy specification for XML under OFGAC framework. Then, we apply the OFGAC approach in two directions: view-based and RDBMS-based.



(a) Combined lattice of DOM and PAR



(b) Combined lattice of SIGN and PAR

Figure. 5: Combination of lattices

A. Observation-based Access Control Policy Specification

Most of the existing approaches on fine grained access control for XML are based on the basic policy specification introduced by Damiani et al. [9] that specifies the access authorization by a 5-tuple of the form $\langle$*Subject*, *Object*, *Action*, *Sign*, *Type*$\rangle$. The "*Subject*" represents the identifiers or locations of the access requests to be granted or rejected. It is denoted by a 3-tuple $\langle UG, IP, SN\rangle$ where UG, IP and SN are the set of user-groups/user-identifiers, the set of completely-specified/patterns-of IP addresses and the set of completely-specified/patterns-of symbolic names respectively. For instance, $\langle$ Physicians, 159.101.*.*, *.hospital.com $\rangle$ represents a subject belonging to the group physicians, issuing queries from the IP address matching with the pattern 159.101.*.* in the domain matching with symbolic name pattern *.hospital.com. The "*Object*" represents the Uniform Resource Identifier (URI) of the elements or attributes in the documents. The URI is specified by the conditional or unconditional path expressions. The "*Action*" is either "read" or "write" or both being authorized or forbidden. The "*Sign*" $\in \{+, -\}$ is the sign of authorization. Sign "+" indicates "allow access", whereas sign "-" indicates "forbid access". The "*Type*" of the access represents the level of access (DTD level or instance level), whether access is applicable only to the local element or applicable recursively to all sub-elements, hard or soft etc. The priority of the type of accesses from highest to lowest are: LDH (Local Hard Authorization), RDH (Recursive Hard Authorization), L (Local Authorization), R (Recursive Authorization), LD (Local Authorization specified at DTD level), RD (Recursive Authorization specified at DTD level), LS (Local Soft Authorization), RS (Recursive Soft Authorization). Since this specification provides users only two choices in accessing the information: either "allow" or "forbid", we call it *Binary-based FGAC Policy* for XML.

In contrast to binary-based FGAC Policy, we specify the Observation-based Access Control Policy for XML under OFGAC framework by a 5-tuple of the form $\langle$*Subject*, *Object*, *Action*, *Abstraction*, *Type*$\rangle$. The components "*Subject*", "*Object*", "*Action*" and "*Type*" are defined exactly in the same way as in case of FGAC policy specification. The component "*Abstraction*" is defined by the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, and α_x and γ_x represent the corresponding abstraction and concretization functions.

Since the "*Object*" represents either XML element or attribute, the following two cases may arise when "*Abstraction*" is applied on them:

- The "*Object*" represents an intermediate element and "*Type*" is "Recursive" (denoted by "R"). In this case, the abstraction defined in the rule for an element is propagated downwards and applied to all its sub-elements and attributes recursively.
- The "*Object*" represents an attribute and "*Type*" is "Local" (denoted by "L"). In this case, only the attribute value is abstracted by following the Galois Connection specified in the rule.

Example 7 illustrates the observation-based access control policy specification for the XML document of Figure 1.

Example 7 *Consider the XML code in Figure 1. The observation-based access control policy under OFGAC framework can be specified as shown in Table 8, where the abstraction functions are defined as follows:*

$$\alpha_{CardNo}(\{d_i \,:\, i \in [1 \ldots 16]\}) = {*}{*}{*}{*}\ {*}{*}{*}{*}\ {*}{*}{*}{*}\ d_{13}d_{14}d_{15}d_{16}$$

$$\alpha_{\top}(X) = \top$$

where X is a set of concrete values and $\top$ is the top most element of the corresponding abstract lattice. The functions α_{iban}, γ_{iban}, γ_{CardNo}, $\gamma_{\top}$ are also defined in this way depending on the corresponding domains. Observe that the identity function id is used to provide the public accessibility of non-sensitive information, whereas the functions $\alpha_{\top}$ and $\gamma_{\top}$ are used to provide private accessibility of highly sensitive information by abstracting them with top most element $\top$ of the corresponding abstract lattice.

B. OFGAC Approaches

Given a binary-based access control policy p or an observation-based access control policy *op* for XML documents, the FGAC and OFGAC can be implemented in two ways:

Table 8: Observation-based Access Control Policy Specification for XML code

Rule	*Subject*	*Object*	*Action*	*Abstraction*	*Type*
R1	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ PersInfo	read	$(\wp(D_x^{con}), id, id, \wp(D_x^{con}))$	R
R2	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ IBAN	read	$(\wp(D_{iban}^{con}), \alpha_{iban}, \gamma_{iban}, D_{iban}^{abs})$	L
R3	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ type	read	$(\wp(D_{type}^{con}), id, id, \wp(D_{type}^{con}))$	L
R4	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ amount	read	$(\wp(D_{amount}^{con}), \alpha_{\top}, \gamma_{\top}, \{\top\})$	L
R5	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ CardNo	read	$(\wp(D_{CardNo}^{con}), \alpha_{CardNo}, \gamma_{CardNo}, D_{CardNo}^{abs})$	L
R6	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ ExpiryDate	read	$(\wp(D_{ExDate}^{con}), \alpha_{\top}, \gamma_{\top}, \{\top\})$	L
R7	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ SecretNo	read	$(\wp(D_{SecNo}^{con}), \alpha_{\top}, \gamma_{\top}, \{\top\})$	L

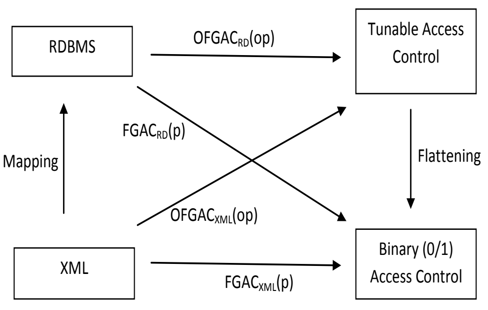


Figure. 6: FGAC Vs. OFGAC

- Non-deterministic Finite Automata (NFA)-based: By applying *p* or *op* directly to the XML documents (view-based) or by rewriting users' XML queries by pruning the unauthorized part in it.
- RDBMS-based: By taking the support of RDBMS, where the XML documents and the XML policies (*p* or *op*) are first mapped into the underlying relational databases and the policy SQL respectively, and then the users' XML queries are mapped into equivalent SQL queries and evaluated on those relational databases by satisfying the policy SQL.

Figure 6 depicts a pictorial representation of these approaches. Observe that the application of FGAC *w.r.t.* *p* results into a binary-based access control system that yields two extreme views to the information: either "allow" or "forbid", whereas the application of OFGAC *w.r.t.* *op*, on the other hand, results into a tunable access control system where partial view of the information at various levels of abstractions is provided. We now discuss the OFGAC approach for XML documents in two directions: view-based and RDBMS-based.

View-based OFGAC for XML. Consider the XML code in Figure 1 and the associated observation-based access control policy specification depicted in Table 8. We know that in view-based approaches for each subject interacting with the system, separate views are generated with respect to the access rules associated with the subject [9]. Therefore, in our example, the XML view corresponding to the users belonging to "customer-care" section of the bank is depicted in Figure 7. The queries issued by a user are then executed on the corresponding secure view without worrying about the security constraint. Consider the following XML query Q_{xml} issued by a personnel in the customer-care section:

```
<?xml version="1.0"? >
<BankCusomers>
<Customer>
<PersInfo>
<Cid> 140062 </Cid>
<Name> John Smith </Name>
<Address>
<street> Via Pasini 62 </street>
<city> Venezia </city>
<country> Italy </country>
<pin> 30175 </pin>
</Address>
<PhoneNo> +39 3897745774 </PhoneNo>
</PersInfo>
<AccountInfo>
<IBAN> IT*** ***** ***** ************ </IBAN>
<type> Savings </type>
<amount> ⊤ </amount>
</AccountInfo >
<CreditCardInfo>
<CardNo> **** **** **** 7835 </CardNo>
<ExpiryDate> ⊤ </ExpiryDate>
<SecretNo> ⊤ </SecretNo>
</CreditCardInfo>
</Customer>
</BankCusomers>
```

Figure. 7: View generated for the employees in bank's customer-care section

Q_{xml} = `/BankCusomers/Customer/AccountInfo[@type=‘‘Savings’’]`

The execution of Q_{xml} on the view of Figure 7 returns the following results:

```
<AccountInfo>
<IBAN> IT*** ***** ***** ************ </IBAN>
<type> Savings </type>
<amount> ⊤ </amount>
</AccountInfo>
```

RDBMS-based OFGAC for XML. Consider the XML document in Figure 1 and the observation-based policy specification in Table 8. By following [18], we first map

Table 9: The equivalent relational database representation of the XML code

(a) *"BankCustomers"*

id	*pid*	*rule*
BC1	null	-

(b) *"Customer"*

id	*pid*	*rule*
C1	BC1	-

(c) *"PersInfo"*

id	*pid*	*rule*
PI1	C1	R1

(d) *"AccountInfo"*

id	*pid*	*rule*
AI1	C1	-

(e) *"CreditCardInfo"*

id	*pid*	*rule*
CI1	C1	-

(f) *"IBAN"*

id	*pid*	*val*	*rule*
IB1	AI1	IT10G 02006 02003 000011115996	R2

(g) *"type"*

id	*pid*	*val*	*rule*
TP1	AI1	Savings	R3

(h) *"amount"*

id	*pid*	*val*	*rule*
AM1	AI1	5000	R4

(i) *"CardNo"*

id	*pid*	*val*	*rule*
CN1	CI1	4023 4581 8419 7835	R5

(j) *"ExpiryDate"*

id	*pid*	*val*	*rule*
EX1	CI1	12/15	R6

the XML document into relational database representation, partially shown in Table 9. Observe that we do not translate the XML policies into the equivalent SQL statements, rather we put the rules into the relational database itself by associating them with the corresponding elements or attributes. The empty rule in a row specifies that the corresponding element (and its subelements and child-attributes) or attribute has public authorization. If any access-conflict occurs for any subelement, it is resolved simply by adopting *abstraction-take-precedence* policy according to which authorization corresponding to more abstract view overrides the authorization corresponding to less abstract view. The users' XML queries are then mapped into SQL representation and are evaluated on this relational database under OFGAC framework as described before. Suppose the following XML query Q_{xml} is issued by an employee from customer-care section of the bank:

Q_{xml} =`/BankCusomers/Customer/AccountInfo[@type=`
`‘‘Savings’’]/IBAN`

Since the OFGAC Policies and XML documents are now in the form of relational database, the system translates Q_{xml} into an equivalent SQL query Q_{rdb} as follows:

Q_{rdb} =
```
SELECT Ch_No.val FROM IBAN Ch_No, type Ch_Tp, AccountInfo
    P_AccInfo, Customer P_Cust, BankCustomers P_BCust WHERE
    (Ch_No.pid=P_AccInfo.id AND Ch_Tp.pid=P_AccInfo.id AND
    Ch_Tp.val=‘‘Savings’’) AND P_AccInfo.pid=P_Cust.id AND
    P_Cust.pid=P_BCust.id
```

The execution of Q_{rdb} on the database of Table 9, by following the OFGAC technique for RDBMS, yields the following result:

val
IT*** ***** ***** ************

Observe that RDBMS-based approaches suffer from time-inefficiency, whereas view-based approaches, on the other hand, suffer from space-inefficiency. The possibility of collusion attacks for XML documents under OFGAC framework is same as described before in case of RDBMS.

V. Related Works

The existing schemes on FGAC for RDBMS suggest to mask the confidential information by special symbols like NULL [20] or $Type$-1/$Type$-2 variables [28], or to execute the queries over the operational relations [26, 32] or authorized views [23, 16], etc.
Wang et al. [28] proposed a formal notion of correctness in fine-grained database access control. They showed why the existing approaches [20] fall short in some circumstances with respect to soundness and security requirements, like when queries contain negation operations. Moreover, they proposed a labeling approach for masking unauthorized information by using two types of special variables (Type-1 or Type-2) as well as a secure and sound query evaluation algorithm in case of cell-level disclosure policies.
In [26, 32], the authors observed that the proposed algorithm in [28] is unable to satisfy the soundness property for the queries containing the negation operations NOT IN or NOT EXISTS. They proposed an enforcing rule to control the information leakage where the query is executed on an operational relation rather than the original relation. However, although the algorithm for Enforcing Rule satisfies the soundness and security properties for all SQL queries, it would not reach the maximum property [28].
The authors in [3] expressed the secret information by an existentially quantified boolean query. They presented a formal model of secret information disclosure that defines a query to be suspicious if and only if the disclosed secret could be inferred from its answer.
Agrawal et al. [1] introduced the syntax of a fine grained restriction command at column level, row level, or cell level. The enforcement algorithm automatically combines the restrictions relevant to individual queries annotated with purpose and recipient information, and transforms the users' queries into equivalent queries over a dynamic view that implements the restriction.
In [31], the authors extended the SQL language to express the FGAC security policies. They provide syntax to create a new policy type or replace the old policy with a new one. Many policy instances of a policy type can be created when needed. Moreover it specifies the operations on the objects that the security policy restricts and the filter list that specifies the data to be accessed in the specific objects. Finally it has constraint expressions whose truth value determine whether the policy will be executed or not.

Rizvi et al. in [23] described two models for fine-grained access control: Truman and Non-Truman models. Both models support authorization-transparent querying. Unlike the Truman model, the Non-Truman model avoids the pitfalls of the query modification approach and allows a great deal of flexibility in authorization, such as authorization of aggregate results. They defined the notions of unconditional and conditional validity of the query, and presented several inference rules for validity.

Kabra et al. [16] defined the circumstances when a query plan is safe with respect to user defined functions (UDFs) and other unsafe functions (USFs). They proposed techniques to generate safe query plans. However, these safe query plans may yield to un-optimized plans.

The authors in [17] presented two models to solve the information leakage problem occurring during query aggregation. The first model is the base model that uses a single inference dispersion value (Δ) for each user where as the second model uses multiple inference dispersion values for each user with a view to provide more accessibility. Whenever a user queries the database, the query is passed through the inference interpreter. Based on the data items already sent to the user and the data items currently requested the interpreter determines if there is a possibility of inference. The interpreter rejects the query if it finds that inference is possible; otherwise, the query is processed. The interpreter determines inference mathematically by using a mechanism called aggregation graphs and setting up a threshold called inference dispersion.

In [15], authors presented a quantitative model for privacy protection. In the model, a formal representation of the user's information states is given, and they estimate the value of information for the user by considering a specific user model. Under the user model, the privacy protection task is to ensure that the user cannot profit from obtaining the private information.

Various proposals in support of fine-grained XML access control have been introduced in the literature. These include View-based [2, 8, 9], Non-Deterministic Finite Automata (NFA)-based [4, 21, 22], RDBMS-based [18, 19, 27] XML Access Control Enforcement Techniques, etc.

The idea of view-based access control is to create and maintain separate view for each user based on the authorization rules. During run-time, users' queries are executed on the corresponding view, without worrying about the security enforcement. However, for a system with large number of users, the view based approach suffers from high maintenance and storage cost, although views are prepared offline. Yu et al. [30] strictly improved the time and space complexity of view-based approaches by taking advantage of structural locality of accessibility where data items grouped together with similar accessibility properties and a Compressed Accessibility Map (CAM) is built that acts as an accessibility index. In addition to the existing authorization privileges ("allow" or "forbid"), Wu and Raun [29] included one more access privilege, called delegatable administrative privilege, where authorization of an element can be propagated to its parent/child elements if the element has delegatable administrative privilege and there is no authorization for its parent/child elements at all. Moreover, any conflict in the access authorization, if occurs, are resolved by assigning priority to each authorization privileges.

The proposals in [22, 21, 4] are based on rewriting the users queries conforming the access control rules by using Non-Deterministic Finite Automata (NFA). The static analysis in [22] compares the schema automata, access-control automata, and query regular expressions, and classifies queries to be either entirely authorized or entirely prohibited before submitting it to the XML engine. For partially authorized XML queries it relies on the XML engine to filter out the data nodes that users do not have authorization to access, *i.e.*, when static analysis cannot provide determinate answers, the scheme relies on run-time checking. QFilter [4] first generates NFA for the set of access control rules. The input query is then processed against the NFA to determine whether the query satisfies the access control rules completely or partially, and accordingly, it is rewritten into a filtered query by removing the conflicting portions from the input query. The proposal in [21] uses NFA to process streaming XML data for access control. Although, Relational Database Management System, due to its structured nature, becomes inappropriate in the context of World Wide Web, most data for XML documents still reside in relational databases behind the scene. The proposals in [18, 27, 19] takes advantage of relational model, by mapping all the XML data and access controls rules for XML data (in XML format) into the equivalent relational database and structured query language representation. Finally, the accessibility of the data are checked at the level of relational database representation using the SQL representation of the rules.

VI. Conclusions

In this paper, we introduced an Observation-based Fine Grained Access Control (OFGAC) framework on top of the traditional FGAC where the confidential information in the database are abstracted by their observable properties and the external observers are able to see this partial or abstract view of the confidential information rather than their exact contents. The traditional FGAC can be seen as a special case of our OFGAC, where the confidential information are abstracted by the top element in the corresponding abstract lattices.

Acknowledgments

Work partially supported by PRIN 2010-11 project "Security Horizons".

References

[1] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. Ex-

tending relational database systems to automatically enforce privacy policies. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pages 1013–1022, Tokyo, Japan, April 5–8 2005. IEEE Computer Society.

[2] Elisa Bertino and Elena Ferrari. Secure and selective dissemination of xml documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002.

[3] Stefan Bottcher, Rita Hartel, and Matthias Kirschner. Detecting suspicious relational database queries. In *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES '08)*, pages 771–778, Barcelona, Spain, March 4–7 2008. IEEE Computer Society.

[4] Luc Bouganim, François Dang Ngoc, and Philippe Pucheral. Client-based access control management for xml documents. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '04)*, pages 84–95, Toronto, Canada, August 31–September 3 2004. Morgan Kaufmann Publishers Inc.

[5] Agostino Cortesi and Raju Halder. Abstract interpretation of recursive queries. In *Proc. of the 9th Int. Conf. on Distributed Computing and Internet Technologies*, pages 157–170. Springer LNCS 7753, 2013.

[6] Agostino Cortesi and Matteo Zanioli. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures*, 37:24–42, 2011.

[7] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252, Los Angeles, CA, USA, January 17–19 1977. ACM Press.

[8] Ernesto Damiani, Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Design and implementation of an access control processor for xml documents. *Journal of computer and telecommunications netowrking*, 33(1–6):59–75, 2000.

[9] Ernesto Damiani, Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security*, 5(2):169–202, 2002.

[10] Raju Halder and Agostino Cortesi. Observation-based fine grained access control for xml documents. In *Proceedings of the 10th International Conference on Computer Information Systems and Industrial Management Applications (CISIM '11)*, pages 267–276, Kolkata, India, December 14–16 2011. Springer CCIS, Volume 245.

[11] Raju Halder and Agostino Cortesi. Abstract interpretation of database query languages. *Computer Languages, Systems & Structures*, 38:123–157, 2012.

[12] Raju Halder and Agostino Cortesi. Fine grained access control for relational databases by abstract interpretation. In Josè Cordeiro, Maria Virvou, and Boris Shishkov, editors, *Software and Data Technologies*, pages 235–249. Springer CCIS, Volume 170, 2012.

[13] Raju Halder and Agostino Cortesi. Abstract program slicing of database query languages. In *Proceedings of the the 28th Symposium On Applied Computing - Special Track on Database Theory, Technology, and Applications*, pages 838–845, Coimbra, Portugal, 2013. ACM Press.

[14] Raju Halder, Shantanu Pal, and Agostino Cortesi. Watermarking techniques for relational databases: Survey, classification and comparison. *Journal of Universal Computer Science*, 16(21):3164–3190, 2010.

[15] Tsan-sheng Hsu, Churn-Jung Liau, Da-Wei Wang, and Jeremy K.-P. Chen. Quantifying privacy leakage through answering database queries. In *Proceedings of the 5th International Conference on Information Security (ISC '02)*, pages 162–176, London, UK, September 30–October 2 2002. Springer LNCS, Volume 2433.

[16] Govind Kabra, Ravishankar Ramamurthy, and S. Sudarshan. Redundancy and information leakage in fine-grained access control. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*, pages 133–144, Chicago, IL, USA, June 27–29 2006. ACM Press.

[17] Sastry Konduri, Brajendra Panda, and Wing-Ning Li. Monitoring information leakage during query aggregation. In *Proceedings of the 4th International Conference in Distributed Computing and Internet Technology (ICDCIT'07)*, pages 89–96, Bangalore, India, December 17–20 2007. Springer LNCS, Volume 4882.

[18] Lazaros Koromilas, George Chinis, Irini Fundulaki, and Sotiris Ioannidis. Controlling access to xml documents over xml native and relational databases. In *Proceedings of the 6th VLDB Workshop on Secure Data Management (SDM '09)*, pages 122–141, Lyon, France, August 28 2009. Springer LNCS, Volume 5776.

[19] Dongwon Lee, Wang-Chien Lee, and Peng Liu. Supporting xml security models using relational databases: A vision. In *Proceedings of the 1st International XML Database Symposium (Xsym '03)*, pages 267–281, Berlin, Germany, September 8 2003. Springer LNCS, Volume 2824.

[20] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases.

In *Proceedings of the 30th international conference on Very Large Data Bases (VLDB '04)*, pages 108–119, Toronto, Canada, August 31–September 3 2004. Morgan Kaufmann Publishers Inc.

[21] Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. Qfilter: fine-grained run-time xml access control via nfa-based query rewriting. In *Proceedings of the 13th ACM International Conference on Information and knowledge management (CIKM '04)*, pages 543–552, Washington D.C., USA, November 8–13 2004. ACM Press.

[22] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. Xml access control using static analysis. *ACM Transactions on Information and System Security*, 9(3):292–324, 2006.

[23] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pages 551–562, Paris, France, June 13–18 2004. ACM Press.

[24] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in Communications*, 21(1):5–19, 2003.

[25] Jie Shi and Hong Zhu. A fine-grained access control model for relational databases. *Journal of Zhejiang University - Science C*, 11(8):575–586, 2010. Zhejiang University Press, co-published with Springer.

[26] Jie Shi, Hong Zhu, Ge Fu, and Tao Jiang. On the soundness property for sql queries of fine-grained access control in dbmss. In *Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS '09)*, pages 469–474, Shanghai, China, June 1–3 2009. IEEE Compueter Society.

[27] Kian-Lee Tan, Mong Li Lee, and Wang Wang. Access control of xml documents in relational database systems. In *Proceedings of the International Conference on Internet Computing (IC '01)*, pages 185–191, Las Vegas, Nevada, USA, June 25–28 2001. CSREA Press.

[28] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the 33rd international conference on Very large data bases (VLDB '07)*, pages 555–566, Vienna, Austria, September 23–27 2007. VLDB Endowment.

[29] Jing Wu, Jennifer Seberry, Yi Mu, and Chun Ruan. Delegatable access control for fine-grained xml. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS '05)*, pages 270–274, Fuduoka, Japan, July 20–22 2005. IEEE Computer Society.

[30] Ting Yu, Divesh Srivastava, Laks V. S. Lakshmanan, and H. V. Jagadish. Compressed accessibility map: efficient access control for xml. In *Proceedings of the 28th international conference on Very Large Data Bases (VLDB '02)*, pages 478–489, Hong Kong, China, August 20–23 2002. VLDB Endowment.

[31] Hong Zhu and Kevin Lu. Fine-grained access control for database management systems. In *Proceedings of the 24th British National Conference on Databases*, pages 215–223, Glasgow, UK, July 3–5 2007. Springer LNCS, Volume 4587.

[32] Hong Zhu, Jie Shi, Yuanzhen Wang, and Yucai Feng. Controlling information leakage of fine-grained access model in dbmss. In *Proceedings of the 9th International Conference on Web-Age Information Management (WAIM '08)*, pages 583–590, Zhangjiajie, China, July 20–22 2008. IEEE Computer Society.