Trust-Based Constraint-Secure Interoperation for Dynamic Mediator-Free Collaboration

Lingli Deng [1,2], Ziyao Xu [1,2], Yeping He [2]
[1] *Graduate University of Chinese Academy of Sciences, Beijing, China*
[2] *Institute of Software, Chinese Academy of Sciences, Beijing, China*
{denglingli,ccxu,yphe}@ercist.iscas.ac.cn

***Abstract*— By collaboration, domains share resources effectively. To maintain security properties of individual domains during collaboration is a key issue. When domains employing heterogeneous RBAC policies collaborate via crossdomain role-role mappings, their locally-defined separation of duty constraints face the risk of breaching. We present the requirements for constraint-secure interoperation, prohibiting implicit authorizations that break constraints from other member domains. We propose a trust-based framework to implement constraint-secure interoperation with differential trust relations between member pairs in open collaborative scenarios. The framework introduces cross-domain migration and remote assurance of constraints to maximize interoperability between mutually trusted domains, ensures separation of constraint conflicts to minimize security risk between distrusted domains. We provide algorithms of a fully distributed implementation, security proofs and demonstrative usage cases for the proposed solution.**

***Index Terms*— Secure Collaboration, RBAC, Separation of Duty, Statically Mutual Exclusive Roles**

I. INTRODUCTION

Emerging Internet-based applications supports *virtual organizations* (such as peer-to-peer networks, distributed database systems) consisting of independent, globally distributed domains (or entities, e.g. individuals or institutions) that share knowledge and resources and work toward a common goal. A domain is a separate, autonomous entity that manages a group of resources with its own independent administration and access control policies [1]. Interopertion provides a means for domains to share resources and services with better performance and resource utility. With the increase in information and data accessibility, there is a growing concern for security and privacy of data, highlighting the need for robust access control management systems, for improperly configured global policy may compromise the security of collaborating domains [2]. In particular, secure interoperation dictates the following principles [2]: (1) **Autonomy** if an access is permitted within an individual domain, it must be permitted under the secure interoperation; and (2) **Security** if an access is not permitted within an individual domain, it must not be permitted under secure interoperation. Even if there exists a trusted mediator in charge of global security administration, it is challenging to achieve secure interoperation [2] [3] [4]. And things get much more complicated in a fully distributed and dynamic environment, where individual domains come and go in an ad hoc way and no trusted mediator is present. In a mediator-free collaboration, a member domain knows little about neither the global system nor the access control policies of other domains, except its own crossdomain relationships. The interdomain integration and mediation have to be distributed to member domains and achieved through their cooperation [1].

Separation of Duty (SoD) is a well-known principle in the field of computer security [5], which prevents fraud and controls error [6]. To complete an n-stepped sensitive task t, an SoD policy requires at least k ($k \leq n$) users be involved. A Statically-implemented SoD (SSoD) policy dictates no k ($k \leq n$) users that together own the permissions to complete t. Due to its inherent richness in modeling a wide range of access control policies [7] and separation of duty [8], Role-Based Access Control (RBAC) provides a promising approach for multidomain collaboration. Statically Mutually Exclusive Roles (SMER) [9] is supported by most RBAC models [10] to implement SSoD policies. Collaborating domains employing RBAC interoperate by means of crossdomain role-role mapping relation, which essentially integrates role hierarchies from member domains into a combined global hierarchy. A crossdomain mapping may lead to constraint violations through its implicit role assignments through membership inheritance along the global hierarchy. Traditional approaches for conflict detection handling this problem within a single domain do not apply in mediator-free environments, for they require global knowledge and are enforced by a central authority.

In this paper, we focus on how to securely ensure the SMER constraints through distributed conflict detection by member domains for a dynamic mediator-free collaboration where no global knowledge or mediation is present. Our contributions can be summarized as follows: (1) we identify the *remote constraint assurance* problem in a mediator-free collaboration, and introduce the concept of *constraint-secure interoperation* to address it; (2) we ex-

This paper is based on "Enforcing Separation of Duty in Ad Hoc Collaboration," by L. Deng, Y. He, and Z. Xu, which appeared in the Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS), Zhangjiajie, China, November 2008. © 2008 IEEE.

This work is supported by the National Natural Science Foundation of China (under Grant No.90818012), the Knowledge Innovation Key Directional Program of Chinese Academy of Sciences (under Grant No.KGCX2-YW-125) and the Municipal Science and Technology Commission of Beijing (under Grant No. Z08000102000801).

tend a local SMER to a globally migrating *MD-SMER* in a multi-domain collaboration, and introduce the concept of *constraint deduction* as the basis for constraint migration and conflict detection; (3) we propose a distributed framework for constraint-secure interoperation with differential trust relations among member domains, where trusted domains cooperate with the constraint originator in migrating and ensuring MD-SMERs, and any mapping implying constraint exposure to distrusted members is denied; (4) we prove the solution's security.

In the following, Section II presents the problem, constraint-secure interoperation, and the basics for our solution. Section III describes the framework with detailed algorithms. Exemplary cases illustrating the framework's usage in practice, and its security proofs appear in Sections IV and V, respectively. Section VI reviews related work. Section VII concludes.

II. Constraint-Secure Interoperation

We assume that all the domains adopt RBAC [8], where permissions are assigned to roles, and users are assigned to roles, thereby acquiring the roles' permissions. The access control policy of a domain is modeled as a tuple $G = \langle U, P, R, UA, PA, RH, SSD\rangle$, including three sets of entities: users(U), permissions(P) and roles(R); two many-to-many relations: user-role assignment (UA) and permission-role assignment (PA); a partial-order role hierarchy (RH)); and a set of SMER constraints (SSD). If $(r_a, r_b) \in RH$ (also written as $r_b \leq_{RH} r_a$), users who are authorized to access r_a (the *senior* role) are authorized to access r_b (the *junior* role). An SMER $s = \langle\{r_1, r_2, ..., r_m\}, t\rangle$ ($0 < t \leq m$), requires that no user in the domain be assigned to more than $t-1$ roles in the conflicting role set $s.c = \{r_1, r_2, ..., r_m\}$.

Given n RBAC domains, $G_i = \langle U_i, P_i, R_i, UA_i, PA_i, RH_i, SSD_i\rangle$, $i = 1, 2, ..., n$, the interoperation is implemented by introducing a set of crossdomain role-role mappings, F, which is selected by member domain administrators through negotiation for collaborative needs to relate roles from different domains. If $(r_a, r_b) \in F$ with $r_a \in R_A$ and $r_b \in R_B$, users who are authorized to access r_a (the *mapping role*) in domain A are authorized to access r_b (the *mapped role*) in domain B, A is called B's *upstream domain*, and B is A's *downstream domain*.

Fig. 1(c) shows a 3-domain interoperation. Alice's book store (Domain A) provides various discount policies:

1) no mail fare for Native Customer (users of A_2),
2) 5% off for Remote Customer (A_3), and
3) special gifts for VIP Customer (A_1).

However, She also dictates that policy 1) and 2) not be accumulated for one customer, which is enforced by an SMER defined in domain A. For daily management, Bob defines three local roles for his private library (Domain B): Manager(B_1), Librarian(B_2) and Reader(B_3). Carl's public library (Domain C) provides differential services to Senior Members(C_1) and Junior Members(C_2), and charges them accordingly. For the purpose of mutual reciprocity, three collaborative relations are to be established:

1) Junior Members of Carl's public library can obtain the Remote Customer's discount from Alice's Book Store (mapping (C_2, A_3));
2) the Librarian of Bob's Private Library can obtain the VIP discount when doing business with Alice's Book Store (mapping (B_2, A_1));
3) through prior payment for collective membership, Readers from Bob's Private Library receive Senior Member class service from Carl's Public Library (mapping (B_3, C_1)).

The only SMER constraint is defined by Alice in domain A as $\langle\{A_2, A_3\}, 2\rangle$ to ensure that "no customer can obtain both discount 1) and 2)".

A. Motivation: Remote Constraint Assurance Problem

Role hierarchy brings implicit effect for the assurance of an SMER. More accurately, constraints are inherited within a role hierarchy [11]. E.g., given the role hierarchy of domain A, where $A_2 \leq_{RH} A_1$ holds, the SMER $s = \langle\{A_2, A_3\}, 2\rangle$ implies a constraint, which is semantically equal to $s' = \langle\{A_1, A_3\}, 2\rangle$ in terms of role A_1, who is a senior of one of the conflicting roles A_2. Because the user who is authorized to access both A_1 and A_3, can attain access to A_2 through its senior A_1. Once s' is violated, s is broken as well. Intuitively, the mutual exclusion against A_3 is inherited by A_1 from its junior A_2. Similarly, the integration of role hierarchies by collaboration brings forward the *remote constraint assurance* problem that a crossdomain mapping may lead to violation to constraints of the downstream domain as they are inherited by roles of the upstream domain.

A newly-added crossdomain mapping (u, v) violates $\langle\{r_1, r_2, ..., r_m\}, t\rangle$, if and only if (1) without (u, v) there is no user in the system who have access to more than $t-1$ roles in $\{r_1, r_2, ..., r_m\}$; and (2) with (u, v) there is at least one such user. The security of an SMER is ensured if each mapping authorized does not violate it. For example, in Fig. 1(a), the only SMER is defined in domain A as $\langle\{A_2, A_3\}, 2\rangle$; and a new domain B is about to join the collaboration by establishing mappings (B_2, A_1) and (B_3, C_1) successively. If (B_3, C_1) is added to Fig. 1(b), as shown by Fig. 1(c), user u violates domain A's SMER $\langle\{A_2, A_3\}, 2\rangle$ by accessing both A_2 and A_3, if (1) u is assigned by B to B_1 or its senior; or (2) u is assigned by B to both B_2 and B_3; or (3) u can access a role mapped to B_1 or its senior through role hierarchies or crossdomain mappings; or (4) u can access two roles which are respectively mapped or senior to B_2 and B_3.

Previous work on secure interoperation provides protection against violations by users within a single domain, leaving the vulnerability for outside users violating local constraints through crossdomain mappings. Hence, we define our goal of constraint-secure interoperation here.

Definition 1 (constraint-secure interoperation): Given a domain $G = \langle U, P, R, UA, PA, RH, SSD\rangle$, a role hierarchy edge $(u, v) \in RH$ ($u, v \in R$) is *constraint-secure* in terms of $\langle UA, RH, SSD\rangle$ if $\forall c \in SSD$, (u, v) does not violate c.

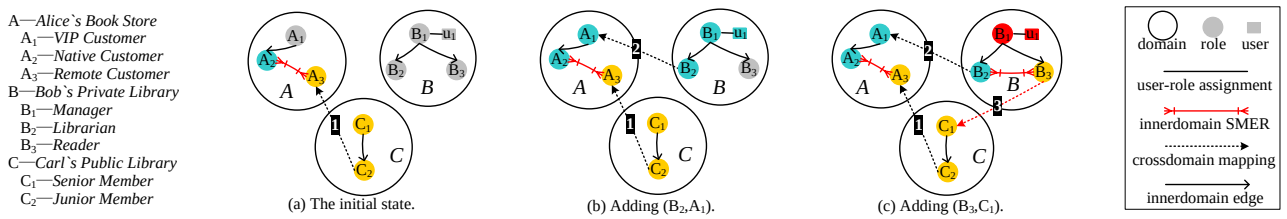

Figure 1. The Remote Constraint Assurance Problem.

Given $G_i = \langle U_i, P_i, R_i, UA_i, PA_i, RH_i, SSD_i \rangle (i = 1...n)$ and a mapping set F, interoperation $Q = \langle \bigcup_{i=1}^{n} U_i, \bigcup_{i=1}^{n} P_i, \bigcup_{i=1}^{n} R_i, UA_Q, \bigcup_{i=1}^{n} PA_i, RH_Q, SSD_Q \rangle (UA_Q = \bigcup_{i=1}^{n} UA_i,\ RH_Q \subseteq \{\bigcup_{i=1}^{n} RH_i \cup F\}\ and\ SSD_Q = \bigcup_{i=1}^{n} SSD_i)$ is *constraint-secure* if $\forall (u, v) \in RH_Q$ and $\forall c \in SSD_Q$, (u, v) does not violate c.

Constraint-secure interoperation explicitly addresses global requirements to protect domains' security in terms of static separation of duty constraints. It is clear that the case demonstrated by Fig. 1(c) does not fulfill the above requirements. The previous work on secure-interoperation in a dynamic mediator-free collaboration environment [1], where every member domain only ensures its own security constraints and only knows about those cross-domain mappings involving itself, cannot guarantee the requirements for constraint-secure interoperation, for the establishment of a crossdomain role-role mapping can influence domains, other than the two directly involved, through transitive mapping relations. For example, in Fig. 1(c) the establishment of (B_3, C_1) leads to a violation by role B_1 in domain B to the SMER constraint of domain A. Since domain B knows nothing about constraints defined in domain A and domain A is unaware of user u from domain B violating its security constraints either, no one can do anything about it.

B. Basic Idea: Constraint Migration

To achieve the goal of constraint-secure interoperation, we devised a crossdomain constraint assurance mechanism for the mediator-free collaboration environment [12]. Before a crossdomain mapping is added, related member domains use the mechanism to assess the intended mapping's influence thoroughly and ensure every one's predefined constraints will not be violated by newly-introduced authorizations. The basic idea is that: each domain disseminates its local SMERs with extended migration information upwards to upstream neighbors; each domain can then check the violation of constraints when adding crossdomain links using the extended SMERs.

Take Fig. 1(b) for example. The SMER defined in A has been disseminated to B_2 and C_2 through (B_2, A_1) and (C_2, A_3), and subsequently through the role hierarchies of B and C to B_1 and C_1. When domain B and C negotiate on the establishment of a new mapping (B_3, C_1), B could tentatively add the mapping, receive the disseminated constraints through it, and analyze newly received constraints with the disseminated constraints from other established mappings as well as its own constraints. B would find out that the very same constraint $\langle \{A_2, A_3\}, 2 \rangle$ would be disseminated from A_3 to another local role B_3, which means, in order to ensure the disseminated constraint, role B_2 and B_3 have to be locally exclusive (no user/role can be authorized to both of them). However, the existence of B_1 as the common senior to both B_2 and B_3 and its assigned user u_1 makes it impossible for domain B to meet the above requirement. Domain B then concludes that (B_3, C_1) violates the disseminated constraint $\langle \{A_2, A_3\}, 2 \rangle$ from domain A through established (B_2, A_1), and denies adding (B_3, C_1) accordingly.

However, for the above constraint migration scheme to work in practice, the domain where a migrating constraint origins must trust the domains, who are on the constraint's migrating path, to honestly disseminate and enforce the constraint. This trust is two-fold, including

1) *trust in honesty*, i.e. the constraint originator trusts others not to manipulate the constraint on migration;
2) *trust in capacity*, i.e. the constraint originator trusts others to possess the ability to correctly understand and enforce the migrated constraint.

Without full two-fold trust among the collaborating domains, the above scheme is hardly useful.

Although it's feasible for individual domains to make sure another domain is trustworthy without centralized mediation, by utilizing Trusted Computing Technologies (TPM and remote attestation protocols) [13]. The reliance on the mutual reciprocal and honest collaboration atmosphere becomes a limitation when cooperating with "semi trusted" domains is desirable, as in more open and competitive environments, where it would be in the interest of a domain to try and violate another domain's SMERs. Therefore, we need to extend the basic framework of [12] to deal with the threats from distrusted member domains.

C. Building Blocks: MD-SMER and EX-SMER

We introduce a partial ordering TR to model the trust relation between collaborating domains, i.e. the trust between two domains is (1) reflexive (a domain always trusts itself); (2) unidirectional (the fact that A trusts B does not necessarily mean that B trusts A); and (3) transitive (A trusts C, if A trusts B and B trusts C). [1]

Definition 2 (trust relation between domains, TR): $TR : S_{Dom} \times S_{Dom}$ is a *partial ordering* relation, such

[1]Our decision to use a partial ordering to model the trust relation is based on its capacity to model differential trust relations, including (1) distrust domains; (2) complete ordering; and (3) lattice-like hierarchies.

TABLE I.
DEFINITIONS OF TOOL FUNCTIONS

Name	Definition
$Dom(r)$	$Dom(r) = d \Leftrightarrow r \in R_d$
$Dom(u)$	$Dom(u) = d \Leftrightarrow u \in U_d$
$Dom(m)$	$Dom(m) = d \Leftrightarrow m.SMER \in SSD_d$
$Ass\_r(u)$	$\{r \in Ass\_r(u) \mid (u,r) \in UA_{Dom(r)}\}$
$Ass\_u(r)$	$\{u \in Ass\_u(r) \mid (u,r) \in UA_{Dom(r)}\}$
$Up(d)$	$\{w \in S_{Dom} \mid \exists (u,v) \in F(u \in R_w, v \in R_d)\}$
$Ups(d)$	$\{w \in S_{Dom} \mid (w \in Up(d)) \vee (\exists (u,v) \in F, d' \in S_{Dom}(u \in R_w, v \in R_{d'}, d' \in Up(d)))\}$
$InLinks(d)$	$\{(u,v) \in F \mid v \in R_d, u \notin R_d\}$
$OutLinks(d)$	$\{(u,v) \in F \mid u \in R_d, v \notin R_d\}$
$InRoles(d)$	$\{r \in R_d \mid \exists (u,r) \in InLinks(d)\}$
$OutRoles(d)$	$\{r \in R_d \mid \exists (r,v) \in OutLinks(d)\}$
$Com(d)$	$\{d' \in S_{Dom} \mid (d' \propto d) \wedge (d \in Ups(d'))\}$

that for any $d, d' \in S_{Dom}$, we say "d trusts d'" or "d' is trusted by d", if $(d, d') \in TR$ (also written as $d \propto d'$); we say "d and d' are mutually trusted", if both $d \propto d'$ and $d' \propto d$ hold (also written as $d \asymp d'$).

Table I lists definitions for tool functions to be used later in this paper: $RH(d)$ returns domain d's role hierarchy; $Dom(r)$, $Dom(u)$ and $Dom(m)$ return the domain where role r and user u resides and MD-SMER m originates; $Ass\_r(u)$ ($Ass\_u(r)$ returns the set of local roles (users) assigned to user u (role r); $Up(d)$ and $Ups(d)$ return the sets of domain d's direct and all upstream domains, respectively; $InLinks(d)$ and $OutLinks(d)$ return domain d's in-coming and out-going cross domain mappings, respectively; and $InRoles(d)$ and $OutRoles(d)$ return domain d's sets of local roles involved in in-coming and out-going mappings, respectively. For a given domain d, $Com(d)$ returns the set of d's downstream domains who trust d, including d itself.

To facilitate the constraint migration and conflict detection, we wrap SMERs with necessary migration information to be used later by a remote domain. Given a role r in a multi-domain collaboration employing RBAC, its *Multi-Domain SMER (MD-SMER)* $m = \langle id(s), r_f, r_t, bmap, c \rangle$ represents an SMER s coming from other domains through a crossdomain mapping, or a transformed SMER of the local domain. The fields are carefully designed to aid the constraint's migration across domains, while keeping the constraint's content from being disclosed across domains which may indicate a breach of its originator's security/confidentiality: s is the original SMER constraint, written as $s = m.SMER$; $id(s)$ stands for the global identification of the migrating constraint, which is locally computed by the originating domain using a predefined well-known function[2]; r_f records the last role outside the local domain where it comes from; r_t states the very first role inside the local domain it resides in; and $bmap$ is an m-bit binary string recording what roles out of $\{r_1, r_2, ..., r_m\}$ (defined by s) are accessible to the current role r, and $m.c = s.t$ is the limit defined by s. For MD-SMERs transformed from local SMERs, their r_f and r_t fields are set to $Null$.

[2]A possible implementation is: $id(s) = SHA-1(s.id|d.id)$, where d is the originating domain of s and $s.id$ is the local id for s in d.

TABLE II.
EXEMPLARY MD-SMER CONSTRAINTS

Role	Link	MD-SMER	EX-SMER
A_2	–	$\langle id_x, -, -, 10, 2 \rangle$	$\langle id_x, A_1, B_2, 10, 2 \rangle$
A_3	–	$\langle id_x, -, -, 01, 2 \rangle$	$\langle id_x, A_3, C_2, 01, 2 \rangle$
A_1	(A_1, A_2)	$\langle id_x, -, -, 10, 2 \rangle$	$\langle id_x, A_1, B_2, 10, 2 \rangle$
B_2	(B_2, A_1)	$\langle id_x, A_1, B_2, 10, 2 \rangle$	-
B_3	–	–	-
B_1	(B_1, B_2)	$\langle id_x, A_1, B_2, 10, 2 \rangle$	-
C_2	(C_2, A_3)	$\langle id_x, A_3, C_2, 01, 2 \rangle$	-
C_1	(C_1, C_2)	$\langle id_x, A_3, C_2, 01, 2 \rangle$	-

Take Fig. 1(b) as an example, supposing the three domains are mutually trusted, Table II gives each role's correspondent MD-SMER for SMER $\langle \{A_2, A_3\}, 2 \rangle$ in domain A. In Table II, the symbol "-" represents $Null$, and the "Link" column lists the MD-SMER's last step of dissemination. For roles inside domain A where the original SMER resides, the MD-SMERs they perceive are transformed locally, so their r_f and r_t fields are $Null$. Through domain A's role hierarchy edge (A_1, A_2), $\langle id_x, -, -, 10, 2 \rangle$ migrates from A_2 to A_1. Similarly, through crossdomain mapping (B_2, A_1), $\langle id_x, -, -, 10, 2 \rangle$ is disseminated from A_1 in A to B_2 in B and transformed into $\langle id_x, A_1, B_2, 10, 2 \rangle$ in order to record its migrating path, so that domain B can tell later that it was disseminated from neighboring domain A through (B_2, A_1).

Definition 3 (constraint access path): The set of MD-SMERs that are currently migrated to role r defines r's *constraint access path*, denoted by $r.cs$. A user u's *constraint access path* is defined as the union of its assigned roles' access paths, i.e. $u.cs = \cup_{r \in Ass\_r(u)} r.cs$.

Definition 4 (constraint exposure set): An MD-SMER m is *exposed*, if there exist roles r and r' such that $e \in r.cs$ and $Dom(e)$ distrusts $Dom(r')$ and r is accessible to r' under the current collaboration. An EX-SMER e is identical to its correspondent exposed MD-SMER s except that $(e.r_t, e.r_f)$ records the next mapping in e's exposure path. The set of EX-SMERs currently migrated to role r defines r's *constraint exposure set*, denoted by $r.os$. The union of local roles' constraint exposure set defines a domain d's *constraint exposure set*, $d.os$.

Take Fig. 1(b) as an example, the rightmost column of Table II gives each role's EX-SMER, assuming the three involved domains distrust each other.

In the following, to further define constraint-secure access paths and conflict-free exposure sets, we define the hierarchy relation between MD-SMERs, the deduction operation on a constraint set, and constraint conflict.

Definition 5 (MD-SMER Hierarchy, MH): Relation $MH : S_{MD} \times S_{MD}$ is a partial ordering, such that for any $x, y \in S_{MD}$, $(x, y) \in MH$, denoted by $x \preceq_{MH} y$, means x is implicitly inferred by y. Formally,[3]

$$MH = \{(x,y) | (x, y \in S_{MD}) \wedge (x.id = y.id) \wedge (x.bmap \odot y.bmap = y.bmap)\}.$$

[3]S_{MD} denotes the set of potential $MD-SMER$s, $\odot$ stands for the binary OR operation.

The MH relation between two MD-SMERs sharing the same SMER origin is inherently the set inclusion relation between the sets of reachable roles recorded in the two's $bmap$ fields. As an EX-SMER is inherently a specialized MD-SMER, the same relation exists for EX-SMERs. If $x \preceq_{MH} y$, the set $\{x, y\}$ is semantically equal to the set $\{x\}$. Like the role hierarchy formed by the partial ordering RH, there is a constraint hierarchy formed by MH. We can alternatively state the fact that "$x \preceq_{MH} y$" by saying "*x is a junior to y*" or "*y is a senior to x*".

Definition 6 (MD-SMER deduction): $MD-SMERs$ x and y are *deductible*, if there exists $z \in S_{MD}$ that is the least common senior to x and y in MH. z is called the *deduction* of x and y, denoted by $z = [x, y]$. $X \subseteq S_{MD}$ is a *deducted set*, if no pair of its members are deductible, denoted by $X \in R_{MD-SMER}$. Formally,

$$\forall x, y, w \in S_{MD}((x \preceq_{MH} [x, y]) \wedge (y \preceq_{MH} [x, y]) \wedge$$

$$((x \preceq_{MH} w) \wedge (y \preceq_{MH} w) \rightarrow [x, y] \preceq_{MH} w)).$$

$$X \notin R_{MD-SMER} \Leftrightarrow (\exists x, y \in X(\exists [x, y] \in S_{MD})).$$

Definition 7 (MD-SMER conflict): An MD-SMER x *contains conflict*, if the number of reachable conflicting roles recorded in $x.bmap$ exceeds the limit $x.c$, written as $\bigtriangledown x$. A deducted MD-SMER set X *contains conflict*, written as $\bigtriangledown X$, if there exists $x \in X$ that contains conflict. Formally,[4]

$$\forall x \in S_{MD}(\bigtriangledown x \Leftrightarrow count([x, y].bmap) \geq x.c).$$

$$\forall X \subseteq S_{MD}(\bigtriangledown X \Leftrightarrow \exists x \in X(\bigtriangledown x)).$$

Definition 8 (constraint-secure access path): A role's (user's) constraint access path is *constraint-secure* in terms of domain d, if it contains no conflicting MD-SMER originating from d. A *(globally) constraint-secure access path* is constraint-secure in terms of every domain.

As later proved by Lemma 3 and Lemma 6, a conflict in $u.cs$ indicates a violation to the correspondent SMER constraint by user u's current authorization state; and a conflict in $d.os$ indicates an undesirable delegation of remote assurance responsibility for correspondent SMER defined by d, to domains distrusted by d. In other words, the requirement for constraint-secure interoperation is effectively satisfied in the presence of heterogeneous trust relations among domains, if and only if "any user u's constraint access path be constraint-secure and any domain d's constraint exposure set be conflict-free", which is to be used by local decision makers to collectively ensure the constraint-security of the global collaboration.

In our distributed framework, a role's constraint access path, which is the accumulated set of disseminated constraints from other domains, disseminates reversely along role hierarchies and crossdomain mappings. A user's constraint access path $u.cs$ is intended to provide two kinds of information to the decision maker when an administrative request for establishing a new cross-domain mapping is presented: first, it contains all the local/remote SMER constraints that have influence on u's further authorizations; second, it records for each SMER constraint the set of conflicting roles which u have access to. In order words, it provides all the information a local domain administrator needs to know when determining whether the user's current authorization state violates related constraints, i.e., whether the user's access path is *constraint-secure* (Definition 8).

Meanwhile, each domain maintains its exposed constraint set, which is intended to record all the locally defined or remote conflicting roles that are accessible to domains distrusted by their originator under the current collaboration. In other words, it provides all the information a local domain administrator needs to know when determining whether its distrusted rivals (domains) collectively have the authorization to violate the SMERs locally or migrated in the form of MD-SMERs, i.e., whether its constraint exposure set contains *conflict*.

III. The Framework

In this section we present a distributed, dynamic trust-based solution to the constraint-secure interoperation problem. In our scheme, established mappings are used to make decisions about subsequent collaboration requests. The idea of using collaboration history in controlling subsequent mapping establishment is inspired by the Chinese Wall security policy [14] in using a user's access history to control its further accesses. MD-SMERs are disseminated recursively along the innerdomain role hierarchy edges and crossdomain mappings to roles (residing in trusted domains) who can access correspondent conflicting roles. The constraint access path of a user u in a member domain represents the partial state of conflicting roles in collaborating domains to which u have access, and the constraint exposure set of a member domain d records the conflicting roles that are currently accessible to other domains distrusted by d. A subsequent request for establishing a new mapping for a trusted domain is approved only when no conflict is introduced in related users' constraint access paths, while a request for a mapping to a distrusted domain is approved only when no conflict is introduced in related domains' constraint exposure sets.

A. Framework Overview

The framework can be represented by a three-layered model (Fig. 2), including (1) the Request Handling Module, which is in charge of generating out-going requests for adding or deleting crossdomain mappings and evaluating in-coming requests from neighbor domains; (2) the Conflict Detection Module, which conducts deduction and detects conflict on constraint sets (access paths or exposure sets); and (3) the Constraint Migration Module, which is responsible for disseminating roles' constraint access paths/exposure sets across domains or within the local domain. Each member domain is equipped with these modules and cooperates with one another to ensure the fulfillment of constraint-secure interoperation.

[4]Function $count$ returns the number of "1" in the input binary string.

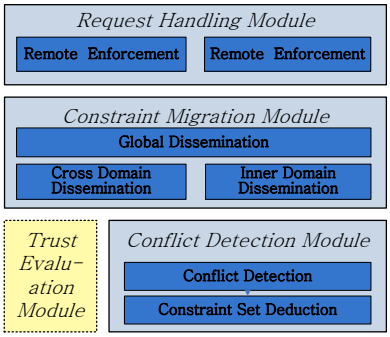

Figure 2. Framework's layered model.

The concepts of constraint-secure access path and conflict-free exposure set are at the heart of our framework: the Constraint Migration Module maintains and updates them according to the established role hierarchies, crossdomain mappings and the trust relations involved; the Conflict Detection Module checks for any conflict in them, and provides the results to the Request Handling Module; and the Requesting Handling Module bases its decision to grant or reject the establishment of a requested mapping on whether they are constraint-security/conflict-free.

B. The Conflict Detection Module

The Conflict Detection Module computes the deducted constraint sets for disseminated/expected constraint access paths or exposure sets (in the form of MD-SMER/EX-SMER sets), and checks if there is any constraint conflict within the resultant constraint sets. These results are to be used by the Request Handling Module to make decisions to grant or reject a new mapping.

$md\_deduction$ given in Table III uses the set union operation to compute the deducted set for a MD-SMER/EX-SMER set. Dependent on the input value of $flag$, the deduction can be partially done only for those constraints sharing the same (r_t, r_f) pair. Function $conflict$ is used to check for conflict within a deducted constraint set. $domain\_conflict$ and $domain\_exposure$ are called by the local domain to check whether the current system state results in any conflict in view of disseminated MD-SMERs and locally exposed EX-SMERs, respectively.

C. The Constraint Migration Module

The Constraint Migration Module is composed of three components: (1) *innerdomain migration* for transforming and disseminating constraints within a single domain through the role hierarchy (e.g. from A_2 to A_1 in Fig. 1(a)); (2) *crossdomain migration* for handling constraint migration/revocation through crossdomain mappings between neighboring domains (e.g. from A_1 to B_2 in Fig. 1(b)); and (3) *global migration* for MD-SMERs' global dissemination recursively among trusted upstream domains and EX-SMERs' global exposure recursively among trustee downstream domains when adding/removing a crossdomain mapping. In the following, we explain each sub-module, whose detailed algorithms are given by Table IV.

1) Innerdomain Migration: The role hierarchy in each domain, adds roles not explicitly used in but senior to the ones in the constraint's definition to an SMER's realm. Two functions are used to compute the MD-SMER for a senior role in its way of dissemination reversely along the role hierarchy edges: $inner\_disseminate$ computes a senior role p's access path to disseminate newly updated junior role $c.cs$ in the local role hierarchy. In opposition, $inner\_disseminate\_r$ removes the constraints coming from junior $c.cs$ out of senior p 's access path. Similar processes for EX-SMER exposure are implemented by $inner\_expose$ and $inner\_expose\_r$, except that the exposure set disseminates in the opposite direction, i.e. from the senior role p to the junior role c.

2) Crossdomain Migration: This sub-module computes (1) updated constraint access path(s) for related roles after a mapping from a trusted domain is added or deleted; and (2)updated constraint exposure(s) for related domains after a mapping from a distrusted domain is added or deleted. Compared with innerdomain migration, which only modifies the $bmap$ of a migrating constraint, crossdomain migration updates a migrating constraint's r_f and r_t as well to record its dissemination/exposure path. Specifically, $external\_disseminate$ takes the newly updated mapping (t, f) and the two roles' current constraint sets to compute updated $t.cs$; given the related roles' current constraint access paths and the role hierarchy of t's domain, function $external\_disseminate\_r$ is designed for updating $t.cs$ by removing constraints from $f.cs$. [5] In case of potential conflict exposure (i.e. there exists some MD-SMER $m \in f.cs$ such that $Dom(m)$ distrusts $Dom(t)$), $external\_disseminate(t, f)$ does not add m to $t.cs$.

Similar procedures for crossdomain exposure set update to $f.os$ according to newly updated $t.os$ are implemented by $external\_expose$ and $external\_expose\_r$.

3) Global Migration: As shown by Fig. 1, the establishment/removal of a crossdomain mapping has profound influence in the collaboration environment, which is not confined to the two directly involved domains. In fact, domain y trusted by another domain x has to ensure the security of MD-SMERs migrated from domain x, if some roles in y have direct or indirect access to roles in x. Therefore, we need to provide algorithms to update all the roles' MD-SMERs and the downstream domains' constraint exposure sets affected by a specific mapping based on the single-step crossdomain dissemination algorithms. Since MD-SMER/EX-SMER migrates upwards/downwards along mappings, constraint/exposure update in one domain may result in subsequent updates in upstream/downstream domains. These recursive processes can be implemented by $recursive\_dissminate$ and $recursive\_expose$, respectively.

[5] Both $inner\_disseminate\_r$ and $external\_dissminate\_r$ are used by $recursive\_disseminate\_r$ in recursively eliminating the effect of an outdated mapping for any upstream domains.

TABLE III.
FUNCTIONS FOR DEDUCTION AND CONFLICT DETECTION.

Function: $md\_deduction(CS, flag)$
Desc: Return the deducted CS.
1 if $(CS = \varnothing)$ return $\varnothing$;
2 $CS' = CS$; $CS'' = \varnothing$;
3 while $(CS' != \varnothing)$ do
4 $find = 0$; $cs' = CS'[0]$;
5 for every cs'' *in* CS'' do
6 if $((cs'.id = cs''.id) AND ((flag = 1) OR$
7 $((cs'.r_f = cs''.r_f) AND (cs'.r_t = cs''.r_t))))$
8 $find = 1$; $cs''.bmap| = cs'.bmap$; break;
9 if $(find = 0)$ $CS'' += cs'$;
10 $CS' -= cs'$;
11 return CS'';

Function: $conflict(CS)$
Desc: Check for conflict in constraint set CS.
1 $CS' = md\_deduction(CS, 1)$;
2 for every $cs \in CS'$ do
3 if$(count(cs.bmap) \geq cs.t)$ return $true$;
4 return $false$;

Function: $domain\_conflict(d)$
Desc: Check for migration conflict in domain d
1 if $(OutLinks(d) = \varnothing)$ return $false$;
2 for every $r \in R_d$ do
3 for every $x \in OutRoles(d)$ do
4 $x.cs = inner\_disseminate(RH_d, x, r)$;
5 if $((conflict(x.cs)) AND (Ass\_u(r) \neq \varnothing))$
6 return $true$;
7 for every $u \in U_d$ do
8 if $(conflict(\bigcup_{r \in Ass\_r(u)} r.cs))$ return $true$;
9 return $false$;

Function: $domain\_exposure(d)$
Desc: Check for exposure conflict in domain d.
1 if $(InLinks(d) = \varnothing)$ return $false$;
2 $xset = \varnothing$;
3 for every $x \in InRoles(d)$ do
4 $xset \cup = x.os$;
5 if $conflict(xset)$ return $true$;
6 return $false$;

Given a newly added (t, f), $recursive\_disseminate$ updates the constraint access paths of those roles in $Ups(Dom(t))$ that are affected by the dissemination of $f.cs$, starting from the mapping role t. Conversely, given a newly deleted mapping (t, f), $recursive\_disseminate\_r$ updates the roles affected by the removal of $f.cs$'s dissemination. Note that when $recursive\_disseminate\_r$ is first called by the Request Handling Module for the $delLink$ operation (see Line 11 of $del\_migration$ in Table VII below), the value of $first$ is always set to 1. When it is called recursively by itself $first$ is always 0. The function checks with its $first$'s value, if it is 0, the current mapping (t, f) handled is not the one being deleted, but one indirectly affected by another mapping's removal. In this case, $t.cs$ needs to be updated to keep consistent with updated $f.cs$. Therefore, $recursive\_disseminate\_r$ first calls $external\_disseminate\_r$ to delete from $t.cs$ constraints migrated through (t, f) earlier, and then calls $external\_disseminate$ to update $t.cs$ with new $f.cs$.

Similarly, $recursive\_expose$ and $recursive\_expose\_r$ update all the related downstream domains' constraint exposure sets in the opposite direction, when adding and deleting a mapping (t, f).

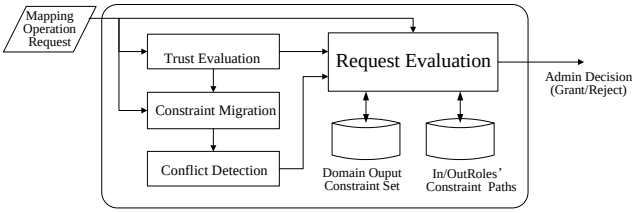

Figure 3. The Request Handling Process.

TABLE V.
ADMINISTRATIVE OPERATION SEMANTICS (CONSTRAINT MIGRATION)

Operation	Semantics
$addLink(t, f)$	$add\_migration(f, t)$
$delLink(t, f)$	$del\_migration(f, t)$
$init(d)$	$\forall r \in InRoles(d)$ $r.os \leftarrow \varnothing$ $(r.cs \leftarrow initial\_disseminate(RH_d, SSD_d, r)$

TABLE VI.
PRECONDITIONS FOR ADMINISTRATIVE OPERATIONS

Operation	Preconditions
$addLink$ (t, f)	$\forall d \in Ups(d), d' \in Com(d)$ $(\neg domain\_conflict(d)) \wedge (\neg domain\_exposure(d'))$

D. The Request Handling Module

Here we modify the semantics and preconditions for related administrative operations in a member domain to ensure constraint-security. Figure 3 shows the interaction between the different modules, when the constraint-security of an administrative request for a crossdomain mapping operation is being evaluated.

As member domains join or leave a dynamic mediator-free environment in an ad hoc way, it is applicable to assume that (1) all the administrative operations (i.e., establishment and removal) on cross domain mappings take place when the related domain joins or leaves the collaboration; and (2) once a member domain joins the interoperation (with all the related mappings established) its access control policy remains unchanged until it finally leaves the interoperation (with all related mappings deleted). Therefore, we only need to consider three kinds of requests in the Request Handling Module: (1) adding a new mapping, i.e. $addLink(t, f)$ in Table VII) invoked by $Dom(f)$; (2) deleting an outdated mapping, i.e. $delLink(t, f)$ invoked by $Dom(f)$; and (3) $init(d)$ for a new domain d to do initialization before joining in.

For the purpose of constraint-secure interoperation, each $addLink$ operation is examined against the preconditions given in Table VI, only those meeting the conditions are committed to successful completion. Two recursive constraint migration procedures described by algorithms of Table VII are used to handle $addLink$ and $delLink$ requests. Each of them composes of two sub-procedures for MD-SMER dissemination upwards into $Upds(Dom(t))$ (Line 11)) and EX-SMER exposure downwards into $Com(Dom(f))$ (Line 1-10). It then uses

TABLE IV.
FUNCTIONS FOR CONSTRAINT MIGRATION.

Function:$inner\_disseminate(RH, c, p)$
Input:c–junior role; p–senior role.
Output:1–if $p.cs$ updated; 0–otherwise.
Desc: Update $p.cs$ for locally updated $c.cs$.
1 if $((c \leq_{RH} p) AND (c.cs \neq \varnothing))$
2 $bak = p.cs$;
3 $p.cs = md\_deduction(p.cs \cup c.cs, 0)$;
4 if $(p.cs \neq bak)$ return $true$;
5 return $false$;

Function:$inner\_expose(RH, p, c)$
Input: c–junior role; p–senior role.
Output:1–if $c.os$ updated; 0–otherwise.
Desc: Update $c.os$ for locally updated $p.os$.
1 if $((c \leq_{RH} p) AND (p.os \neq \varnothing))$
2 $bak = c.os$;
3 $c.os = md\_deduction(p.os \cup c.os, 0)$;
4 if $(c.os \neq bak)$ return $true$;
5 return $false$;

Function:$inner\_disseminate\_r(RH, c, p)$
Desc: Update $p.cs$ for locally updated $c.cs$.
1 if $((c \leq_{RH} p) AND (c.cs \neq \varnothing))$
2 $bak = p.cs$;
3 $p.cs- = c.cs$; //remove those from $c.cs$
4 if $(p.cs \neq bak)$ return $true$;
5 return $false$;

Function: $inner\_expose\_r(RH, p, c)$
Desc: Update $c.os$ for locally updated $p.os$.
1 if $((c \leq_{RH} p) AND (p.os \neq \varnothing))$
2 $bak = c.os$;
3 $c.os- = p.os$; // remove those from $p.os$
4 if $(c.os \neq bak)$ return $true$;
5 return $false$;

Function:$external\_disseminate(RH, f, t)$
Input:f–mapped role; t–mapping role.
Output:1–if $t.cs$ updated; 0–otherwise.
Desc: Update $t.cs$ for remotely updated $f.cs$.
1 if $(f.cs \neq \varnothing)$
2 $bak = t.cs$; $addset = f.cs$;
3 for every $m \in f.cs$ do
4 if $(Dom(m) distrusts Dom(t))$
5 $addset- = m$;
6 for every $m \in addset$ do
7 $m.r_f = f$; $m.r_t = t$;
8 $t.cs = md\_deduction(addset \cup t.cs, 0)$;
9 if $(t.cs \neq bak)$ return $true$;
10 return $false$;

Function:$external\_expose(RH, t, f)$
Input: f–mapped role; t–mapping role.
Output:1–if $f.os$ updated; 0–otherwise.
Desc: Update $f.os$ for remotely updated $t.os$.
1 if $(t.os \neq \varnothing)$
2 $bak = f.os$; $addset = t.os$;
3 for every $e \in t.os$ do
if $(Dom(e) distrusts Dom(f))$
5 $addset- = e$;
6 for every $e \in addset$ do
7 $e.r_f = f$; $e.r_t = t$;
8 $f.os = md\_deduction(addset \cup f.os, 0)$;
9 if $(f.os \neq bak)$ return $true$;
10 return $false$;

Function:$external\_disseminate\_r(RH, f, t)$
Desc: Update $t.cs$ for remotely updated $f.cs$.
1 if $(f.cs \neq \varnothing)$
2 $bak = t.cs$;
3 for every $m \in t.cs$ do
4 if $((m.r_f = f) AND (m.r_t = t))$
5 $t.cs- = m$; //remove those from $f.cs$
6 if $(t.cs \neq bak)$ return $true$;
7 return $false$;

Function: $external\_expose\_r(RH, t, f)$
Desc: Update $f.os$ for remotely updated $t.os$.
1 if $(t.os \neq \varnothing)$
2 $bak = f.os$;
3 for every $e \in f.os$ do
4 if $((e.r_f = f) AND (e.r_t = t))$
5 $f.os- = e$; // remove those from $t.os$
6 if $(f.os \neq bak)$ return $true$;
7 return $false$;

Function:$recursive\_disseminate(f, t)$
Input:f–mapped role; t–mapping role.
Desc: Update $t.cs$ recursively in $Ups(Dom(t))$
for remotely updated $f.cs$ (for some $addLink$).
1 if $(external\_disseminate(f, t) = false)$ return;
2 for every $x \in InRoles(Dom(t))$ do
3 if $(inner\_disseminate(RH_{Dom(t)}, t, x) = true)$
4 for every $(u, v) \in InLinks(Dom(t))$ do
5 $recursive\_disseminate(x, u)$;

Function:$recursive\_expose(t, f)$
Input: t–mapped role; t–mapping role.
Desc: Update $f.os$ recursively in $Com(Dom(t))$
for remotely updated $t.os$ (for some $addLink$).
1 if $(external\_expose(t, f) = false)$ return;
2 for every $y \in OutRoles(Dom(f))$ do
3 if $(inner\_expose(RH_{Dom(t)}, y, f) = true)$
4 for every $(y, v) \in OutLinks(Dom(f))$ do
5 $recursive\_expose(y, v)$;

Function:$recursive\_disseminate\_r(f, t)$
Desc: Update $t.cs$ recursively in $Ups(Dom(t))$
for remotely updated $f.cs$ (for some $delLink$).
1 if $(external\_disseminate\_r(f, t) = false)$ return;
2 if $(first = 0)$ $external\_disseminate(f, t)$;
3 for every $x \in InRoles(Dom(t))$ do
4 if $(inner\_disseminate(RH_{Dom(t)}, t, x) = true)$
5 $inner\_disseminate(RH_{Dom(t)}, t, x)$;
6 for every $(u, v) \in InLinks(Dom(t))$ do
7 $recursive\_disseminate\_r(x, u, 0)$;

Function:$recursive\_expose\_r(t, f)$
Desc: Update $f.os$ recursively in $Com(Dom(t))$
for remotely updated $t.os$ (for some $delLink$).
1 if $(external\_expose\_r(t, f) = false)$ return;
2 $external\_expose(t)$;
3 for every $y \in OutRoles(Dom(f))$ do
4 if $(inner\_expose(RH_{Dom(f)}, y, f) = true)$
5 $inner\_expose(RH_{Dom(f)}, y, f)$;
6 for every $(y, v) \in OutLinks(Dom(f))$ do
7 $recursive\_expose\_r(y, v)$;

the precondition given by Table VI to make the decision of whether to accept an $addLink$ request. Together, the collaboration of involved domains distinguishes two cases based on the trust relation of a constraint's originator and the current mapping domain in question and handles them separately: for each MD-SMER $m \in f.cs$, the framework allows $Dom(f)$ to delegate constraint m's assurance responsibility to $Dom(t)$ by the means of constraint migration and remote enforcement (i.e. m is added into $t.cs$), if $Dom(t)$ trusts $Dom(t)$ in terms of honesty and capacity in order to maximize their interoperability; otherwise, the framework protects $Dom(m)$'s security by ensuring no EX-SMER conflict exists in updated $Dom(m).os$ (i.e. domains that are distrusted by domain $Dom(m)$ together own the permissions to violate its local SMERs), because of the lack of trust foundation for constraint migration and remote assurance.

In summary, $addLink(t, f)$ and $delLink(t, f)$ lead to MD-SMER updates in $Dom(t)$ and its upstream domains $Ups(t)$ by calling $recursive\_disseminate(f, t)$

TABLE VII.
FUNCTIONS FOR REQUEST HANDLING.

Function: $add\_migration(t, f)$	**Function:** $del\_migration(t, f)$
Input:(t, f)–newly added mapping.	**Input:**(t, f)–newly deleted mapping.
Desc: Global MD-SMER/EX-SMER update for adding (t, f).	**Desc:**Global MD-SMER/EX-SMER update for deleting (t, f).
1 $ex\_flag = 0$;	1 $ex\_flag = 0$;
2 for every $m \in f.cs$ do	2 for every $m \in f.os$ do
3 if $Dom(m)$ distrusts $Dome(t)$	3 if $((e.r_f = f)AND(e.r_t = t))$
4 $ex\_flag = 1$; $f.cs- = m$;	4 $ex\_flag = 1$; $f.os- = e$;
5 $m.r_f = f$; $m.r_t = t$; $f.os+ = m$;	5 if $(ex\_flag = 1)$
6 if $(ex\_flag = 1)$	6 for every $y \in OutRoles(Dom(f))$
7 for every $y \in OutRoles(Dom(f))$ do	7 if $(inner\_expose\_r(RH_{Dom(f)}, y, f) = true)$
8 if $(inner\_expose(RH_{Dom(f)}, y, f) = true)$	8 $inner\_expose(RH_{Dom(f)}, y, f)$;
9 for every $(y, v) \in OutLinks(Dom(f))$ do	9 for every $(y, v) \in OutLinks(Dom(f))$ do
10 $recursive\_expose(y, v)$;	10 $recursive\_expose\_r(y, v)$;
11 $recusrive\_disseminate(f, t)$;	11 $recursive\_disseminate\_r(f, t, 1)$;

and $recursive\_disseminate\_r(f, t, 1)$, respectively; also, they would lead to constraint exposure updates in $Dom(f)$ and its downstream domains by calling $recursive\_expose(t, f)$ and $recursive\_expose\_r(t, f)$, respectively. And for a new coming domain d, operation $init(d)$ initializes d's constraint exposure set and access paths of its in-coming roles from the local SMER set SSD_d by setting $d.os$ to $\varnothing$ and calling $initial\_disseminate$.

Given the set of crossdomain mappings to be added, a member domain joins a constraint-secure interoperation by taking the following two steps: first, it executes the $init$ operation given in Table V to transform its locally defined SMERs into MD-SMERs; second, it cooperates with other domains to add expected mappings by executing a series of $addLink$ operations. Similarly, to take departure from the current constraint-secure collaboration, a domain executes a series of $delLink$ operations, respectively.

IV. CASE STUDY

We use the example in Fig. 1 to illustrate the process of constraints' migration and assurance in the framework. Domain A defines an SMER $\langle\{A_2, A_3\}, 2\rangle$. After the $init(A)$ operation, $A_1.cs = \{\langle id_x, -, -, 10, 2\rangle\}$. The following three examples differ in the trust relations among member domains.

Example 1: A, B and C are mutually trusted. (i.e. Exposure update functions all return $false$.)

1.1: $addLink(C_2, A_3)$**.** $recursive\_disseminate$ is called by domain A for (C_2, A_3). It first calls $external\_disseminate$ to disseminate $A_3.cs$ to C_2, resulting in $C_2.cs = \{\langle id_x, A_1, B_2, 01, 2\rangle, \langle id_y, A_1, B_2, 10, 2\rangle\}$. $recursive\_disseminate$ returns as $InLinks(C) = \varnothing$, and the update procedure is over. Domain C then calls $domain\_conflict$. Because $OutRoles(C) = \{C_2\}$, $inner\_disseminate$ instances are called to disseminate updated $C_2.cs$ to C_1, and $C_1.cs = C_2.cs$. (C_2, A_3) is added successfully as $domain\_conflict$ returns $false$.

1.2: $addLink(B_2, A_1)$**.** We assume before joining, A does the innerdomain dissemination of MD-SMERs for A_1 by calling $inner\_disseminate$ when executing $init(A)$, and $A_1.cs = \{\langle id_x, -, -, 10, 2\rangle, \langle id_y, D_3, A_3, 10, 2\rangle\}$. $recursive\_disseminate$ is called by A for (B_2, A_1). The function first calls $external\_disseminate$ to disseminate $A_1.cs$ to role B_2. As a result, $B_2.cs$ is updated to $\{\langle id_x, A_1, B_2, 10, 2\rangle, \langle id_y, A_1, B_2, 10, 2\rangle\}$. $recursive\_disseminate$ returns as $InLinks(B) = \varnothing$, and the update procedure is over. Domain B then calls $domain\_conflict$. Because $OutRoles(B) = \{B_2\}$, $inner\_disseminate$ instances are called to disseminate updated $B_2.cs$ to B_3 and B_1. But B_3 is no senior to B_2, so $B_3.cs$ remains unchanged, while $B_1.cs = B_2.cs$. $domain\_conflict$ returns $false$, so (B_2, A_1) is added successfully.

1.3: $addLink(B_3, C_1)$**.** $recursive\_disseminate$ is called by B for (B_3, C_1). Firstly, $external\_disseminate$ is called to disseminate $C_1.cs$ to B_3, resulting in $B_3.cs = \{\langle id_x, C_1, B_3, 01, 2\rangle, \langle id_y, C_1, B_3, 10, 2\rangle\}$. Because $InLinks(B) = \varnothing$, $recursive\_disseminate$ terminates. Domain B then uses $domain\_conflict$ to check for conflicts. Since $OutRoles(B) = \{B_2, B_3\}$, instances of $inner\_disseminate$ are called to disseminate $B_3.cs$ to B_2 and B_1. $B_2.cs$ is not changed, but $B_1.cs$ becomes to $\{\langle id_x, A_1, B_2, 11, 2\rangle, \langle id_y, A_1, B_3, 10, 2\rangle\}, \langle id_y, C_1, B_2, 10, 2\rangle\}$, containing a conflict. Because $B_1 \in Ass\_r(u_1)$, $domain\_conflict$ returns $true$ for $conflict$ reports $true$ for user u_1. Therefore, adding (B_3, C_1) is denied.

Example 2: A and C are mutually trusted; but they both distrust B.

2.1: $addLink(C_2, A_3)$**.** The same as **1.1** in *Example 1*.

2.2: $addLink(B_2, A_1)$**.** As A distrusts B, it executes $addLink(B_2, A_1)$, which adds $\langle id_x, A_1, B_2, 10, 2\rangle$ into $A_1.os$ and $A_2.os$. A then executes $domain\_exposure$ and finding no conflict in current $A.os = \{\langle id_x, A_1, B_2, 10, 2\rangle\}$. (B_2, A_1) is added successfully.

2.3: $addLink(B_3, C_1)$**.** Since C distrusts B, C executes $addLink(B_3, C_1)$, which adds $\langle id_x, B_3, C_1, 01, 2\rangle$ to $C_1.os$ and calls $recursive\_expose$ which in turn calls $inner\_expose$ to add $\langle id_x, B_3, C_1, 01, 2\rangle$ into $C_2.os$ and $external\_expose$ to add $\langle id_x, C_2, A_3, 01, 2\rangle$ into $A_3.os$, resulting in a conflict. As $domain\_exposure(A)$ returns $true$, adding (B_3, C_1) is denied.

Example 3: A, B and C are mutually distrusted. (i.e. Access path update functions all return $false$.)

3.1: $addLink(C_2, A_3)$**.** Since A distrusts C, A executes $addLink(C_2, A_3)$, which adds $\langle id_x, A_3, C_2, 01, 2\rangle$ into

$A_3.os$. A then executes $domain\_exposure$ and finds no conflict in current $A.os = \{\langle id_x, A_3, C_2, 01, 2\rangle\}$. Since no conflict is detected, (C_2, A_3) is added successfully.

3.2: $addLink(B_2, A_1)$**.** Since A distrusts B, it executes $addLink(B_2, A_1)$, which adds $\langle id_x, A_1, B_2, 10, 2\rangle$ into $A_1.os$. A then executes $domain\_exposure$ and finds conflict in current $A.os$ since $A_3.os = \{\langle id_x, A_3, C_2, 01, 2\rangle\}$. As the precondition for $addLink$ does not hold for $domain\_exposure(A.os)$ returns 1, (B_2, A_1) is denied.

The following observations can be drawn from the above exemplary cases, whose general validity is proved formally in Section V. On one hand, although Example 1 and 2 assume different trust relation between A and B, users from B do not violate the SMER defined by A in both cases: in Example 1 this is done through the collaboration of both A, B and C, while in Example 2 this is achieved merely by the collaboration of A and its trusted partner C, which effectively minimizes the risk of violation against A's SMER incurred by unfounded reliance on distrusted B. On the other hand, comparing Example 3 with 1 and 2, it is clear that by MD-SMER migration and remote assurance delegation to trusted partners, domain A achieves maximum interoperation with domain C (by successful establishment of (C_2, A_3)) as long as there is sound trust basis (i.e. A trusts C).

V. Security Analysis

In this section, we analyze the correctness and completeness of the framework. As a direct application of the security analysis given in [12], we omit the detailed proofs in the following. Using the results of Lemma 1 and Lemma 2, Lemma 3 reveals the correspondence between the conflicts within a role's constraint access path and its potential constraint violation between constraint originator and its trusted domains. Similarly, using the results of Lemma 4 and Lemma 5, Lemma 6 confirms the correspondence between the conflicts within a domain's constraint exposure set and the potential constraint violation by the constraint originator's distrusted domains. And based on them, Theorem 1 proves that under our framework the system state is always secure according to innerdomain SMER constraints and the responsibility for remote constraint assurance is always delegated to trusted domain(s) only, and Theorem 2 proves that a legitimate request for a crossdomain mapping is always granted as long it won't incur any potential violation to innerdomain SMER constraints or reliance on the good behavior of distrusted domain(s).

Lemma 1: If $m \in r.cs$, we have: (1)$Dom(m)$ trusts $Dom(r)$; and (2)the conflicting roles of $m.SMER$ marked "1" in $m.bmap$, are accessible to r under current collaboration.

Lemma 2: If SMER s's conflicting role r_0 is accessible to role r and $Dom(s)$ trusts $Dom(r)$, there exists an MD-SMER $m \in r.cs$ that $m.SMER = s$ with r_0 marked by 1 in $m.bmap$.

Lemma 3: There exists conflict m in user u's constraint access path, if and only if $Dom(m)$ trusts $Dom(u)$ and u violates SMER $m.SMER$ under current collaboration.

Lemma 4: If $e \in Dom(e).os$, the conflicting roles of $e.SMER$ marked "1" in $e.bmap$, are accessible to at least one of $Dom(e)$'s distrusted domains under current collaboration.

Lemma 5: If SMER s's conflicting role r_0 is accessible to role r and $Dom(s)$ distrusts $Dom(r)$, there exists an EX-SMER $e \in Dom(s).os$ such that $e.SMER = s$ and r_0 is marked by 1 in $e.bmap$.

Lemma 6: There exists conflict e in its originator's exposure set $Dome(e).os$, if and only if $Dom(e)$'s distrusted domains collectively have the authorization to violate $e.SMER$ under current collaboration.

Theorem 1 (Correctness): Each mapping added by $addLink$ incurs neither (1)actual violation to any innerdomain SMER by a user residing in a trusted domain; nor (2)potential violation to any innerdomain SMER by distrusted domains even through collusion.

Theorem 2 (Completeness): Each mapping denied by $addLink$ incurs either (1)actual violation to some innerdomain SMER by a user residing in a trusted domain; or (2)potential violation to some innerdomain SMER by distrusted domains even through collusion.

VI. Related work

Gong et al. [2] characterized the principles that must be satisfied to compose a global secure DAC policy for domains employing the HRU model [15]. Their concept of secure interoperation, which is widely accepted by subsequent researchers, dictates to maintain the partial ordering between user groups. Dawson et al. [16] presented a mediator based approach to provide secure interoperability for heterogeneous databases. This approach assumes a mandatory access control policy, such as the Bell LaPadula [17], which is not flexible and not applicable in many commercial applications.

[3] examined the issue of interoperability between two domains employing RBAC, and provided IRBAC2000 model for dynamic role translation. IRBAC2000 considered two security issues: infiltration (unexpected implicit crossdomain authorization) and implicit promotion (role loop in the integrated hierarchy). However, IRBAC2000 does not apply to collaboration of more than two domains, and it does not consider SoD policies, either. Authors of [4] proposed a policy integration framework for merging RBAC policies of multiple domains into a global access control policy, including conflict resolution for a special kind of SMERs (between two roles).

In all such approaches a trusted third party that has a global view of the collaboration environment is required to perform the secure policy composition and integration, and therefore is hardly useful in dynamic collaboration environment with no mediator. Shehab et al. [1] presented a mediator-free collaboration environment, where domains join and leave in an ad hoc manner and no trusted mediator is present. They proposed a distributed secure interoperability framework for such environment,

in which domains collaborate in making localized access control decisions without mediation.

Role-based-access control systems typically have mutually exclusive constraints on the roles a user can assume to implement SSoD policies. While this is a solved problem when working within a single domain, when multiple domains are introduced this constraint can become problematic to ensure, in that a local user can assume the role of a user in a co-operating domain and then "call back" to the original domain under a different role. While that problem has been addressed in [1] we address a harder problem, where users in a co-operating domain that are introduced into ours can assume multiple roles without our being able to control it. By combining the original idea of constraint migration with constraint exposure, we extend the basic solution in [12] for collaboration among mutually-trusted domains and the simple trust-based solution in [18] for communities with binary trust boundaries[6], to accommodate a more competitive and realistic collaboration environment with differential trust relations among member domains. Since the lattice-based trust relation models the scenarios considered by [12] and [18] as two special cases, the trust-based framework presented in this paper generalizes the two earlier solutions.

VII. Conclusion

The paper considers the remote constraint assurance problem in a dynamic mediator-free collaboration among RBAC domains with differential trust relations: how to establish cross-domain role-role mappings in such a way that mutually exclusive constraints are globally ensured without relying on any mediator or third-party. We present the problem, formalize it, and develop a distributed framework, which is composed of a set of layers that are collectively responsible for migrating and ensuring the extended constraints by other trustworthy member domains as well as its originating domain, while keeping them from being exposed to distrusted communities at the same time.

References

[1] M. Shehab, E. Bertino, and A. Ghafoor, "Secure collaboration in mediator-free environments," *Proceedings of 12th ACM Conference on Computer and Communications Security*, pp. 58–67, 2005.

[2] L. Gong and X. Qian, "Computational issues in secure interoperation," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 43–52, 1996.

[3] A. Kapadia, J. Al-Muhtadi, R. Campbell, and D. Mickunas, "IRBAC 2000: secure interoperability using dynamic role translation," *Proceedings of 1st International Conference on Internet Computing*, 2000.

[4] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor, "Secure interoperation in a multidomain environment employing RBAC policies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 11, pp. 1557–1577, 2005.

[5] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of IEEE*, vol. 63, pp. 1278–1308, 1975.

[6] D. Clark and D. Wilson, "A comparison of commercial and military computer security policies," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 184–194, 1987.

[7] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 85–106, 2000.

[8] D. Ferraiolo, D. Kuhn, and R. Chandramouli, *Role-based access controls*. Artech House Boston, 2003.

[9] N. Li, Z. Bizri, and M. Tripunitara, "On mutually-exclusive roles and separation of duty," *Proceedings of 11th ACM Conference on Computer and Communications Security*, pp. 42–51, 2004.

[10] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[11] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[12] L. Deng, Y. He, and Z. Xu, "Enforcing separation of duty in ad hoc collaboration," *Proceedings of 9th International Conference for Young Computer Scientists*, 2008.

[13] R. Sandhu and X. Zhang, "Peer-to-peer access control architecture using trusted computing technology," *Proceedings of 10th ACM Symposium on Access Control Models and Technologies*, pp. 147–158, 2005.

[14] D. Brewer and M. Nash, "The chinese wall security policy," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 206–214, 1989.

[15] M. Harrison, W. Ruzzo, and J. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[16] S. Dawson, S. Qian, and P. Samarati, "Providing security and interoperation of heterogeneous systems," *Distributed and Parallel Databases*, vol. 8, no. 1, pp. 119–145, 2000.

[17] D. Bell and L. LaPadula, "Secure computer systems: mathematical foundations," *Rapport technique*, vol. 2547, 1973.

[18] L. Deng, Y. He, and Z. Xu, "Separation of duty in trust-based collaboration," *Proceedings of 4th International Conference on Information Security and Cryptology*, 2008.

Lingli Deng is currently a Ph.D. candidate at the Graduate University of Chinese Academy of Sciences. She received her BS degree in computer science from the University of Science and Technology of China in 2003. Her research interests include P2P security and distributed access control.

Ziyao Xu is currently a Ph.D. candidate at the Graduate University of Chinese Academy of Sciences. He received his BS degree in computer science from the University of Science and Technology of China in 2003. His research interests include P2P security and trusted computing.

Yeping He received his PhD degree from the Nanjing University of Aeronautics and Astronautics in 1999, his MS and BS degrees in mathematics from the Lanzhou University in 1985 and 1982, respectively.

He is currently a professor of computer science at the Institute of Software, Chinese Academy of Sciences. His current research interests include secure protocol design, system security and trusted computing.

[6]domain collective of mutually trusted members, where all the other domains outside the community boundary are distrusted by the insiders