

Appendix

Theorem 10.1 (Commutativity of RBAC-integrate): The policy integration operation performed by *RBAC-integrate* is commutative.

Proof: RBAC-integrate is commutative if for any two domains A and B, $RBAC-integrate(G_A, G_B) = RBAC-integrate(G_B, G_A)$, where G_A and G_B are the RBAC graphs of domain A and B respectively. The commutativity of *RBAC-integrate* depends on the commutativity of *role-integrate*. Therefore, we first analyze the algorithm *role-integrate*. *Role-integrate* performs role comparison and linking in a recursive manner. Roles are linked by calling *link* function which is symmetric. Linking of equivalent roles (lines 11 - 14 of *role-integrate*) and overlapping roles (lines 27 - 31) is symmetric and hence commutative. For the containment case, assume that $contained(r_B, r_A)$ is true. When $role-integrate(r_A, r_B)$ is called then the code in lines 15 - 18 is executed, and when $role-integrate(r_B, r_A)$ is called, the code in lines 21 - 24 is executed. In both cases, role r_A is split and a junior role r_{Aj} is created with $r_A \geq_I r_{Aj}$, and r_{Aj} is linked to r_B with same permission assignment and junior roles. This implies that the containment case is also symmetric and commutative.

It can be proved using induction that $role-integrate(r_A, r_B)$ and $role-integrate(r_B, r_A)$ produces same number of roles during the process of integration and they have same permission assignment and role-hierarchy. Hence, *role-integrate* is commutative, implying that *RBAC-integrate* is commutative. ■

Theorem 10.2 (Associativity of RBAC-integrate): The policy integration operation performed by *RBAC-integrate* is associative.

Proof: Let G_A , G_B , and G_C be the RBAC graph of domain A, B, and C respectively.

$P = RBAC-integrate(G_A, G_B)$, $Q = RBAC-integrate(G_B, G_C)$, $X = RBAC-integrate(P, G_C)$, $Y = RBAC-integrate(G_A, Q)$

To prove that policy integration operation is associative, we need to prove that the graph X is *isomorphic* to Y. Two policy models are said to be *isomorphic* if there is 1:1 onto correspondence between their elements and they have the same relationships [20]. To show that two final integrated policy models X and Y are isomorphic, we define a morphism $\phi(X \rightarrow Y)$ as follows:

- For a user $u_i \in X$, $\phi(u_i) = u_i$
- For a permission $p_j \in X$, $\phi(p_j) = p_j$
- For a role $r' \in X$, $\phi(r') = r$ such that $pset_{assign}(r') = pset_{assign}(r)$

In order to prove that ϕ is an isomorphism we need to show the following:

- (i) ϕ is 1:1 and onto
- (ii) $R(U) \in R_X$ if and only if $R(\phi(U)) \in R_Y$ (U is a vector).

ϕ is onto: The elements in X and Y can be divided into two types: (i) elements which are present in G_A , G_B , and G_C , (ii) elements that are created in the process of integration of local graphs. As stated in the above theorem that RBAC-integrate satisfies the element preservation property, therefore all the elements of type (i) are present in both X and Y .

Type (ii) elements include those roles that are not present in G_A , G_B , and G_C and are created during the process of policy integration. These roles are created by the role split function in the RBAC-integrate algorithm. Note that type (ii) elements do not include any redundant role as the redundant roles that are created in the policy integration step are eliminated from X and Y . To complete the proof that ϕ is onto, we need to show that for all type (ii) roles $r \in Y$, there exists $r' \in X$ such that $\phi(r') = r$ and for all p such that $p \in pset_{assign}(r) \Rightarrow p \in pset_{assign}(r')$

In the following we use the terminology $r \in dom(X)$ if $r \in G_X$ or r is created by splitting a role $r_s \in dom(X)$. Without loss of generality, assume that there exists a role $r_A \in G_A$ such that $pset(r_A) \supseteq pset(r)$. Also r is created by splitting role r_A i.e., $r \in dom(A)$. Since r is created in the process of integration, therefore one of the following three conditions holds for r .

- a. $\exists r_{BA} \in dom(B): eq_role(r, r_{BA}) \wedge \neg \exists r_{CA} \in dom(C): eq_role(r, r_{CA})$
- b. $\exists r_{CA} \in dom(C): eq_role(r, r_{CA}) \wedge \neg \exists r_{BA} \in dom(B): eq_role(r, r_{BA})$
- c. $\exists r_{BA} \in dom(B), r_{CA} \in dom(C) : eq_role(r, r_{BA}) \wedge eq_role(r, r_{CA})$

Case a: $\exists r_{BA} \in dom(B): eq_role(r, r_{BA}) \wedge \neg \exists r_{CA} \in G_C: eq_role(r, r_{CA})$

The above implies that there is no role in G_C whose permission set overlaps with that of r or r_{BA} . Role r does not exist in Q ; however, r_{BA} may or may not exist in Q .

If r_{BA} exists in Q then $r_{BA} \in G_B$ and the following is true in Y :

$$(i) (r_A \geq r) \wedge (r_A \text{ contains } r_{BA}) \wedge (\neg r_{BA} \text{ contains } r_A)$$

If r_{BA} does not exist in Q , then there exists a role $r_B \in G_B$ such that that $pset(r_B) \cap pset(r_A) = pset(r_{BA})$, and the following hold in Y :

$$(ii) (r_A \geq r) \wedge (r_A \text{ overlaps } r_B)$$

Since $eq_role(r, r_{BA})$ holds, therefore $pset_{assign}(r_{BA}) = pset_{assign}(r)$ and $pset(r_{BA}) = pset(r)$

For the case $r_{BA} \in G_B$ and $r_A \in G_A$, since r_A contains r_{BA} , when integrating G_A and G_B , a role r' junior to r_A is created and is assigned the permission in the set $pset_{assign}(r_{BA}) \cap pset_{assign}(r_A)$. This means that there exists a role r' in P with $pset_{assign}(r') = pset_{assign}(r_{BA}) \cap pset_{assign}(r_A) = pset_{assign}(r_{BA}) = pset_{assign}(r)$. Also, when integrating P with G_C role r' is not split nor the permission in the set $pset_{assign}(r')$ gets redistributed as there is no role in G_C whose permission set overlaps with that of r' .

For the case $r_{BA} \notin G_B$, $r_A \in G_A$ and $r_B \in G_B$, since r_A overlaps r_B , when integrating G_A and G_B , role r' junior to r_A , and r_{BA} junior to r_B are created with $pset_{assign}(r') = pset_{assign}(r_B) = pset_{assign}(r_{BA}) \cap pset_{assign}(r_A)$. This means that there exists a role r' in P with $pset_{assign}(r') = pset_{assign}(r_{BA}) = pset_{assign}(r)$. Also, when integrating P with G_C role r' is not split nor the permission in the set $pset_{assign}(r')$ gets redistributed as there is no role in G_C whose permission set overlaps with that of r' .

Therefore for a type (ii) role $r \in Y$, for which case a holds, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$, i.e., $\varphi(r') = r$. In a similar manner, we can prove the above for cases b and c as well. Hence, for all type (ii) roles $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$, i.e., $\varphi(r') = r$.

Now, we need to show that for all roles $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$. We have proved this for type (ii) roles, now we need to prove it for type (i) roles. Type (i) role can be further classified into two types: (a) roles which remain unsplit during policy integration; (b) roles which split in the policy integration step. Note that the permissions assigned to a role are removed from that role only if it gets split in the process of integration. Consider an unsplit role r in Y and with out loss of generality assume that $r \in G_A$. Since r is an unsplit role therefore, there does not exist any role $r'' \in \{G_B, G_C\}$ such that $pset_{assign}(r) \subset pset_{assign}(r'')$. This and the element preservation property implies that there exists a role $r' \in X$, such that $pset_{assign}(r') = pset_{assign}(r)$.

We need to prove the above for the type (i) roles that get split. Consider a role $r \in Y$ that got split in the process of policy integration to produce a junior role r_j . We already proved that there exists a role $r_j' \in X$ such that $pset_{assign}(r_j') = pset_{assign}(r_j)$. Without loss of generality suppose that $r \in G_A$. Note that $r_j \notin \{G_A, G_B, G_C\}$, which also implies that $r_j' \notin \{G_A, G_B, G_C\}$. Therefore there exists a role r' that produce r_j' after splitting. We maintain that $pset_{assign}(r') = pset_{assign}(r)$. Suppose this is not the case and $pset_{assign}(r') \neq pset_{assign}(r)$. Both r_j and $r_j' \in dom(A)$, which implies that $r' \in dom(A)$. Suppose that $pset_{assign}(r') \supset pset_{assign}(r)$. Note that permissions are removed from a role only if the role gets split and the removed permissions are assigned to the newly created role that is made junior to the role being split. Before splitting, r' and r have same permission assignment. However, after splitting we assume that $pset_{assign}(r') \supset pset_{assign}(r)$, implying that either $pset_{assign}(r_j') \subset pset_{assign}(r_j)$ which is not possible, or r has at least one more newly created junior role r_{j2} which acquires some of the permissions that were earlier assigned to r . If this is the case then r_{j2} must be equivalent to some role $r_{j2'} \in X$ with $pset_{assign}(r_{j2}') = pset_{assign}(r_{j2})$. Nevertheless, r_{j2}' resulted from the split of role r' . This implies that all the permissions in the $pset_{assign}(r') \setminus pset_{assign}(r)$ are removed from r' and are assigned to r_{j2}' . Therefore, $pset_{assign}(r) \not\subset pset_{assign}(r')$

If we assume $pset_{assign}(r') \subset pset_{assign}(r)$ then, either $pset_{assign}(r_j') \supset pset_{assign}(r_j)$ which is not possible; or there exists at least one more newly created child role r_{j2}' ($r_{j2}' \neq r_j'$) of role r' . In this case $pset_{assign}(r_{j2}') = pset_{assign}(r) \setminus pset_{assign}(r')$. Note that $r_{j2}' \in dom(A)$ and therefore there exists a role $r'' \in \{G_B, G_C\}$ such that

either r' contains r'' or r' overlaps r'' . The element preservation property of RBAC-integrate ensures that r'' also exists in Q . When integration between G_A and Q is performed role r is compared with r'' and role r is split to produce a child role r_{j2} with $pset_{assign}(r_{j2}) = pset_{assign}(r) \cap pset_{assign}(r'') = pset_{assign}(r_{j2}')$. This proves that $pset_{assign}(r') \not\subset pset_{assign}(r)$ provided r is split once or twice. Using induction we can prove that $pset_{assign}(r') \not\subset pset_{assign}(r)$ is independent of the number of times role r is split. The above implies that for a type (i) split role $r \in Y$, there exists a role $r' \in X$ such that $pset_{assign}(r') = pset_{assign}(r)$, hence $\varphi(r') = r$.

The final step in proving that φ is onto is to show that all the elements in X map to at least one element in Y . The element preservation property of RBAC-integrate maintains that all the user, permissions and type (i) roles that are present in X are also present in Y . So, all the users, permissions and type (i) roles in X can be mapped to at least one element in Y . Since we disallow non-redundant roles and addition of new permissions and users during the process of integration therefore both X and Y have same number of type (ii) roles. We already proved that for every type (ii) role in Y there exists a type (ii) role in X with the same permission assignment. Since the cardinality of type (ii) roles in both X and Y is same, therefore there exists a 1:1 correspondence between the type two roles in X and Y .

This concludes the proof that φ is onto.

φ is 1:1 (for all $e_1, e_2 \in X$, $\varphi(e_1) = \varphi(e_2) \rightarrow e_1 = e_2$)

The element preservation property of the integration algorithm implies that all the elements in the input graphs G_A, G_B, G_C are present in X and Y . Moreover, RBAC-integrate does not add any new user, permission and type (i) roles, i.e., the cardinality of user set, permission set, and type (i) role set is same in both X and Y . We already proved that φ is onto. Since we disallow non-redundant roles and duplicate permission assignment during the process of integration therefore both X and Y have same number of type (ii) roles. This implies that there is 1:1 correspondence between the user, permission and role elements between X and Y . Hence, φ is 1:1.

Relationship Preservation: To conclude the proof that φ is isomorphic, we need to show that any relation $R(U) \in R_X$ if and only if $R(\varphi(U)) \in R_Y$. The relationship preservation property of RBAC-integrate guarantees that each relation R (except the P -assign) in the input RBAC graph has a corresponding relationship R' in the integrated RBAC graph. We already proved that for any role r' in X , there exists exactly one role r in Y such that that $pset_{assign}(r) = pset_{assign}(\varphi(r))$. Moreover, φ is a 1:1 morphism. This implies that for any permission p , P -assign(r, p) $\in R_X$ if and only P -assign($\varphi(r), p$) $\in R_Y$.

This concludes the proof that φ is isomorphic, implying that the operator RBAC-integrate is associative.

■

Proof of Theorem 7.2: We prove this theorem separately for *role assignment*, *role-specific SoD*, and *user-specific SoD* constraints.

Sub-proof 1: Any state S reachable from multi-domain RBAC graph G is secure with respect to the role-assignment constraint of all collaborating domains. We prove this claim by contradiction. Suppose that the above statement is not true. This means that in some state S reachable from G there exists a user $u_i \in U_k$ who accesses a role $r_j \in R_k$ ($s_{ij} = 1$, $s_{ij} \in \pi_{ur_k}(S)$), while $a_{ij} = 0$, where, $a_{ij} \in \pi_{ur}(A_k^+)$, i.e., there is no intra-domain access path from u_i to r_j . The above implies that in the multi-domain RBAC graph G , there is a path from u_i to r_j that consists of at least two cross-domain edges. Without loss of generality, assume that these cross-domain edges are (r_l, r_m) and (r_n, r_p) , where, $r_l, r_p \in R_k$ and $r_m, r_n \notin R_k$; and $(u_{ir_l} = 1) \wedge (r_m \stackrel{*}{\geq} r_n \vee r_m = r_n) \wedge (r_p \stackrel{*}{\geq} r_j \vee r_p = r_j)$.

Since there is no intra-domain access path from u_i to r_j , $u_{ir_j} = 0$ is specified as one of the constraint to the IP problem (constraint transformation rule 1). Therefore, in any feasible solution $u_{ir_j} = 0$ and $u_{ir_p} = 0$. There are two possibilities for the variable u_{ir_n} in any feasible (optimal feasible) solution: 1) $u_{ir_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *ConfRes* removes the edge (r_n, r_p) . 2) $u_{ir_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{ir_m} = 0$, otherwise it removes the edge (r_n, r_p) .

In either case, any cross-domain edge leading u_i to r_j through r_n is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. Hence in the resulting graph G there is no cross-domain path from u_i to r_j , implying that $s_{ij} = 0$. This contradicts our initial assumption.

Sub-proof 2: Any state S reachable from G is secure with respect to the role-specific SoD constraint of all collaborating domains. We prove this statement by considering all possible role-specific SoD violations that might occur as a result of interoperation. The following cases capture all the role-specific SoD violations in the multi-domain environment:

Case 1: In this case, a local user u_l accesses two conflicting roles r_i and $r_j \in R_k$. There are four sub-cases corresponding to case 1, These sub-cases are shown in Figures 15(a-d).

Sub-case 1(a): The security policy of domain k does not allow u_l to access any of the roles r_i and r_j . If we assume that in some state S , u_l is able to access r_i and r_j through some cross-domain role (see Figure 15(a)), then this will be a violation of role-assignment constraint of domain k . However, all the reachable states from the multi-domain RBAC graph obtained after applying conflict resolution algorithm, *ConfRes*, are secure with respect to the role-assignment constraints of all collaborating domains (proved above). Hence in this sub-case, u_l cannot access r_i and r_j simultaneously.

Sub-case 1(b): RBAC policy of domain k allows u_l to access r_i but not r_j as depicted in Figure 15(b). Since the multi-domain policy is secure with respect to the role-assignment constraints of domain k (proved above),

therefore, u_i cannot access r_j through a cross-domain path, implying that SoD violation between r_i and r_j never occurs in this case.

Sub-case 1(c): Suppose u_i is assigned to r_s and $r_s \geq_A^* r_i$, $r_s \geq_A^* r_j$. Moreover, r_i and r_j are conflicting roles as shown in Figure 15(c). A *role-specific SoD violation* occurs if u_i activates one of the conflicting roles, say r_i , and inherits the other one, say r_j , through r_t such that $(r_s \geq_A^* r_t \vee r_s = r_t) \wedge r_t \geq_I^* r_j$. For a hierarchically consistent RBAC policy, the conflicting role set of a junior role must be contained in the conflicting role set of the senior role. $r_t \geq_I^* r_j \Rightarrow \text{conf-rset}(r_t) \supseteq \text{conf-rset}(r_j)$. This means that $r_i \in \text{conf-rset}(r_t)$. If there is no inter-domain path from u_i to r_t then user u_i cannot access r_t and r_i simultaneously implying that u_i cannot access r_i and r_j simultaneously. If there exists an inter-domain path from u_i to r_t , then by using induction we can show that there exist a role $r_u \in R_k$ such that $(r_s \geq_A^* r_u \vee r_s = r_u) \wedge (r_u \geq_I^* r_t) \wedge \text{conf-role}(r_u, r_i)$ and there does not exist a cross-domain role $r_o \notin R_k$ such that $r_s \geq_I^* r_o \geq_I^* r_u$. If $r_s = r_u$ then this leads to sub-case 1(d) discussed next. If not then this means that u_i cannot access r_u and r_i simultaneously implying that u_i cannot access r_t and r_i simultaneously, which in turns imply that u_i cannot access r_j and r_i simultaneously.

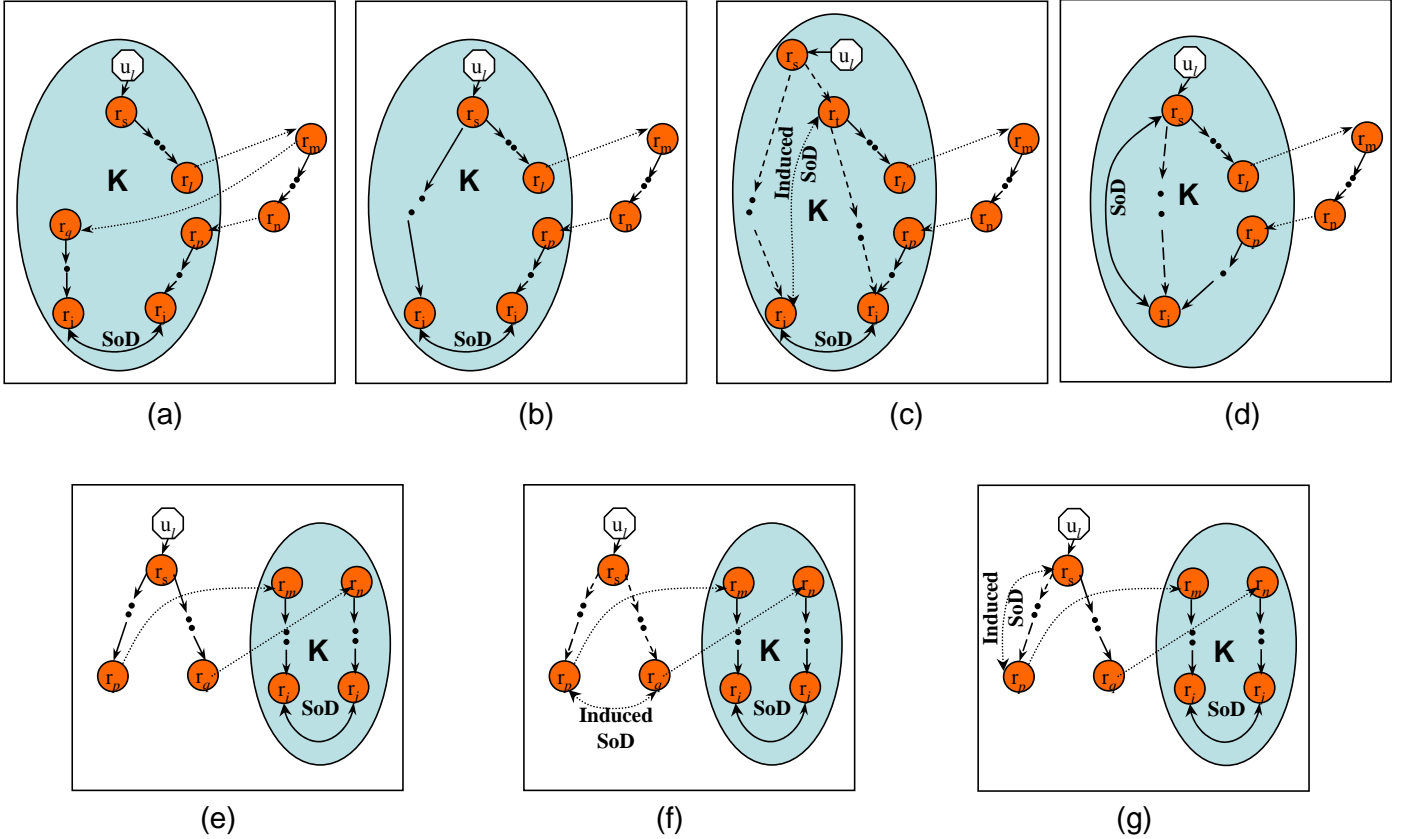


Figure 15. Cases of role-specific *SoD* violations involving cross-domain paths

Sub-case 1(d): Suppose u_l is assigned to r_s and $r_s \geq_A^* r_i$. Moreover, r_s and r_i are activation time conflicting roles as shown in Figure 15(d). If security policy of domain k is consistent then there is no intra-domain path from r_s to r_i consisting of only I -edges. Suppose that there is a cross-domain path from r_s to r_i . Such a path must have at least two cross-domain edges. Without loss of generality, assume that these cross-domain edges are (r_l, r_m) and (r_n, r_p) , where, $r_l, r_p \in R_k$ and $r_m, r_n \notin R_k$; and $\left(r_s \geq_I^* r_l \vee r_s = r_l\right) \wedge \left(r_m \geq_I^* r_n \vee r_m = r_n\right) \wedge \left(r_p \geq_I^* r_i \vee r_p = r_i\right)$. This cross-domain path enables any user to access permissions of r_i by accessing role r_s , which is a violation of SoD constraint between r_s and r_i . At least one user activates role r_s (Step 1 of the *ConfRes* algorithm and transformation rules 3 and 4 ensures that each role in the multi-domain graph is accessed by at least one user). Let the user be u_l . Since r_s and r_i are conflicting roles, therefore $u_{r_s} + u_{r_i} \leq 1$ is one of the constraint of the IP problem formulated in the step 4 of conflict resolution algorithm *Confres*. Since $u_{r_s} = 1$, therefore in any feasible solution $u_{r_i} = 0$ and $u_{r_p} = 0$. There are two possibilities for the variable u_{r_n} in any feasible (optimal feasible) solution:

1. $u_{r_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *Confres* removes the edge (r_n, r_p) .
2. $u_{r_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{r_m} = 0$, otherwise it removes the edge (r_n, r_p) .

In either case, any cross-domain edge leading u_l to r_j through r_n is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. Hence in the resulting graph G there is no cross-domain path from r_s to r_i , implying that u_l cannot access role r_s and r_i simultaneously.

Case 2: In this case, a foreign user $u_l \notin U_k$ accesses two conflicting roles r_i and $r_j \in R_k$. There are three sub-cases corresponding to case 2. Figures 15(e), 15(f) and 15(g) depicts these sub-cases.

Sub-case 2(a): Suppose u_l is assigned to r_s and there is a cross-domain path from r_s to r_i and from r_s to r_j as shown in Figure 15(e). For the cross-domain path from r_s to r_i the following hold:

$$\left(r_s \geq_I^* r_p \vee r_s = r_p\right) \wedge \left(r_p \geq_I^* r_m\right) \wedge \left(r_m \geq_I^* r_i \vee r_m = r_i\right)$$

Similarly, for the cross-domain path from r_s to r_j the following hold:

$$\left(r_s \geq_I^* r_q \vee r_s = r_q\right) \wedge \left(r_q \geq_I^* r_n\right) \wedge \left(r_n \geq_I^* r_j \vee r_n = r_j\right)$$

Since r_i and r_j are conflicting roles and a user u_l assigned to r_s have an access path to both r_i and r_j , therefore $u_{r_i} + u_{r_j} \leq 1$ is one of the constraint of the IP problem formulated in the step 4 of conflict resolution algorithm *Confres*. At least one user activates role r_s (Step 1 of the *ConfRes* algorithm and

transformation rules 3 and 4 ensures that each role in the multi-domain graph is accessed by at least one user). Let the user be u_i , i.e., $u_{r_s} = 1$, which also implies that $u_{r_p} = 1$ and $u_{r_q} = 1$. There are three possibilities for the variables u_{r_i} and u_{r_j} in any feasible solution.

1. $u_{r_i} = 0$ and $u_{r_j} = 0$, implying that $u_{r_m} = 0$ and $u_{r_n} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edges (r_p, r_m) and (r_q, r_n) .
2. $u_{r_i} = 0$ and $u_{r_j} = 1$, implying that $u_{r_m} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edge (r_p, r_m) .
3. $u_{r_i} = 1$ and $u_{r_j} = 0$, implying that $u_{r_n} = 0$. If this is an optimal solution then step 7 of *ConfRes* removes the edge (r_q, r_n) .

In any of the above cases, at least one of the cross-domain paths from r_s to r_i or r_j is removed in the process of conflict resolution. Hence, u_i cannot access both r_i and r_j simultaneously in the resulting RBAC graph G .

Sub-case 2(b): Suppose u_i is assigned to r_s and $r_s \geq_A^* r_p \wedge r_s \geq_A^* r_q$. Let there be a cross-domain path from r_p to r_i and a cross-domain path from r_q to r_j . This is depicted Figure 15(f). These cross-domain relationship $r_p \geq_I^* r_i$ and $r_q \geq_I^* r_j$ induces an *SoD* constraint between r_p and r_q as shown in Figure 15(e). This implies that user u_i cannot activate r_p and r_q concurrently, and therefore cannot access the cross-domain roles r_i and r_j simultaneously.

Sub-case 2(c): Suppose u_i is assigned to r_s and $r_s \geq_A^* r_p \wedge (r_s \geq_I^* r_q \vee r_s = r_q)$. Let there be a cross-domain path from r_p to r_i and a cross-domain path from r_q to r_j . The relation $r_q \geq_I^* r_j$ implies $r_s \geq_I^* r_j$. This is depicted Figure 15(g). These cross-domain relationship $r_p \geq_I^* r_i$ and $r_s \geq_I^* r_j$ induces an *SoD* constraint between r_p and r_s as shown in Figure 15(e). This implies that user u_i cannot activate r_s and r_p concurrently, and therefore cannot access the cross-domain roles r_i and r_j simultaneously.

Any of the *role-specific SoD* constraint can be reduced to one of the above cases. In all of the above cases, we have proved that *SoD* violation between conflicting roles can never happen. Hence, any state S reachable from the multi-domain RBAC graph G obtained after applying conflict resolution algorithm, *ConfRes*, is secure with respect to the *role-specific SoD* constraints of all collaborating domains.

Sub-proof 3: Any state S reachable from G is secure with respect to the user-specific *SoD* constraint of all collaborating domains. A user-specific *SoD* violation of role r_i occurs when a user u_i belonging to the conflicting user set(s) of r_i accesses r_i through multiple paths and at least one of such path includes cross-

domain edges. This is shown in Figure 16, in which users u_1, u_2, \dots, u_m conflict with user u_{dt} for role r_t . The following relationship exists among the roles depicted in Figure 16.

$$\left(r_s \stackrel{*}{\geq}_A r_t \vee r_s = r_t \right) \wedge \left(r_s \stackrel{*}{\geq}_I r_t \vee r_s \stackrel{*}{\geq}_A r_t \vee r_s = r_t \right) \wedge \left(r_l \stackrel{*}{\geq}_I r_m \right) \wedge \left(r_m \stackrel{*}{\geq}_I r_n \right) \wedge \left(r_n \stackrel{*}{\geq}_I r_p \right) \wedge \left(r_p \stackrel{*}{\geq}_I r_t \vee r_p = r_t \right)$$

Where, $r_s, r_l, r_p,$ and $r_t \in R_k$ and $r_m, r_n \notin R_k$, otherwise, domain k 's RBAC policy becomes inconsistent. The case when r_s and r_t are not distinct is trivial and does not involve any cross-domain path for *SoD* violation. The following discussion considers the case when r_s and r_t are distinct roles.

In Figure 16, a user specific *SoD* is violated when u_{dt} activates role r_t and any of the users conflicting with u_{dt} for role r_t accesses role r_l . By accessing role r_l , a user, say u_1 , accesses the permissions of r_t through the cross-domain path.

After step 3 of the conflict resolution algorithm, *ConfRes*, all the user specific *SoD* constraints in the multi-domain RBAC graph G can be reduced to the case shown in Figure 16. Since users u_1, u_2, \dots, u_m conflict with user u_{dt} for role r_t , therefore the following is included as one of the constraints to the IP problem formulated in step 4 of *ConfRes*.

$$\sum_{i=1}^m u_{ir_t} + u_{dtr_t} \leq 1, \text{ Also } u_{dtr_t} \text{ is set to one in step 3 of the algorithm } \textit{Confres}. \text{ This implies that in any}$$

feasible solution the the IP problem, $u_{ir_t} = 0$ for all $i \in \{1, 2, \dots, m\}$.

There are two possibilities for the variable u_{ir_n} in any feasible (optimal feasible) solution:

1. $u_{ir_n} = 1$. If this is an optimal feasible solution to the IP problem, then step 7 of the algorithm *ConfRes* removes the edge (r_n, r_p) .
2. $u_{ir_n} = 0$. If this yields an optimal solution then step 7 of the algorithm *ConfRes* removes the edge (r_l, r_m) if $u_{ir_m} = 0$, otherwise it removes the edge (r_n, r_p) .

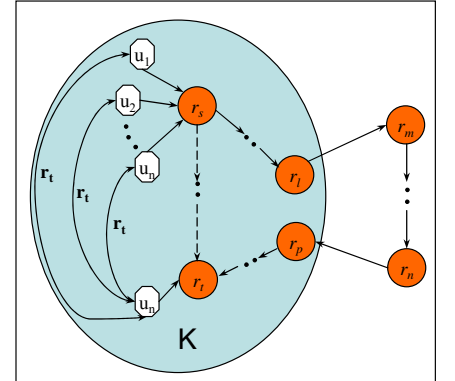


Figure 16. User-specific *SoD* violation through a cross-domain path

In either case, any cross-domain edge leading u_i to r_t through r_n , is dropped. If there are multiple such paths through other cross-domain roles, then in a similar manner those paths will be eliminated by *ConfRes*. This implies that no user u_i belonging to the conflicting user set(s) of r_t can access r_t through a cross-domain path.

Hence, any state S reachable from the multi-domain RBAC graph G obtained after applying conflict resolution algorithm, *ConfRes*, is secure with respect to the user-specific *SoD* constraints of all collaborating domains provided their access control policies are consistent.

This concludes the proof of Theorem 7.2. ■