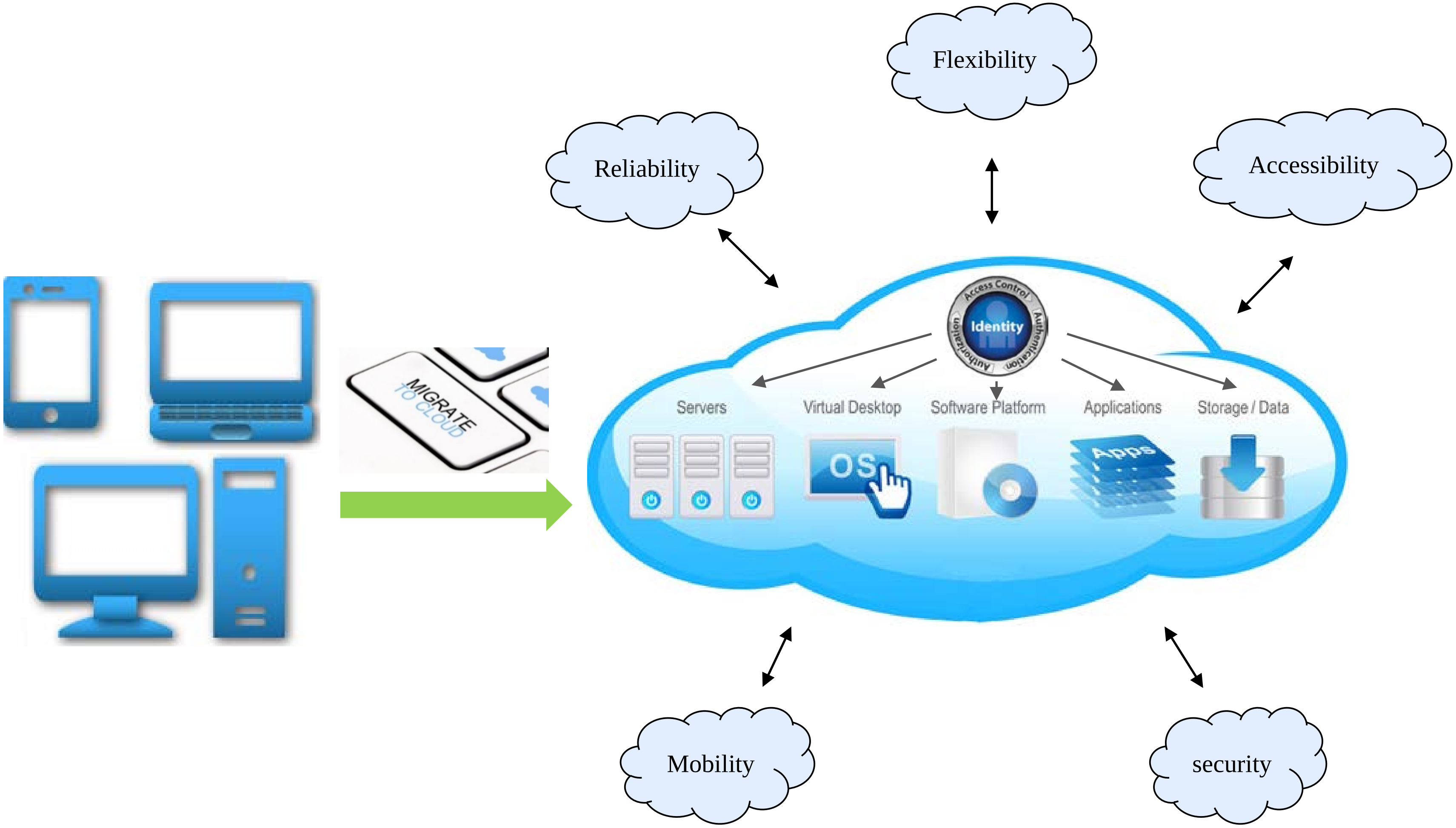
MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust

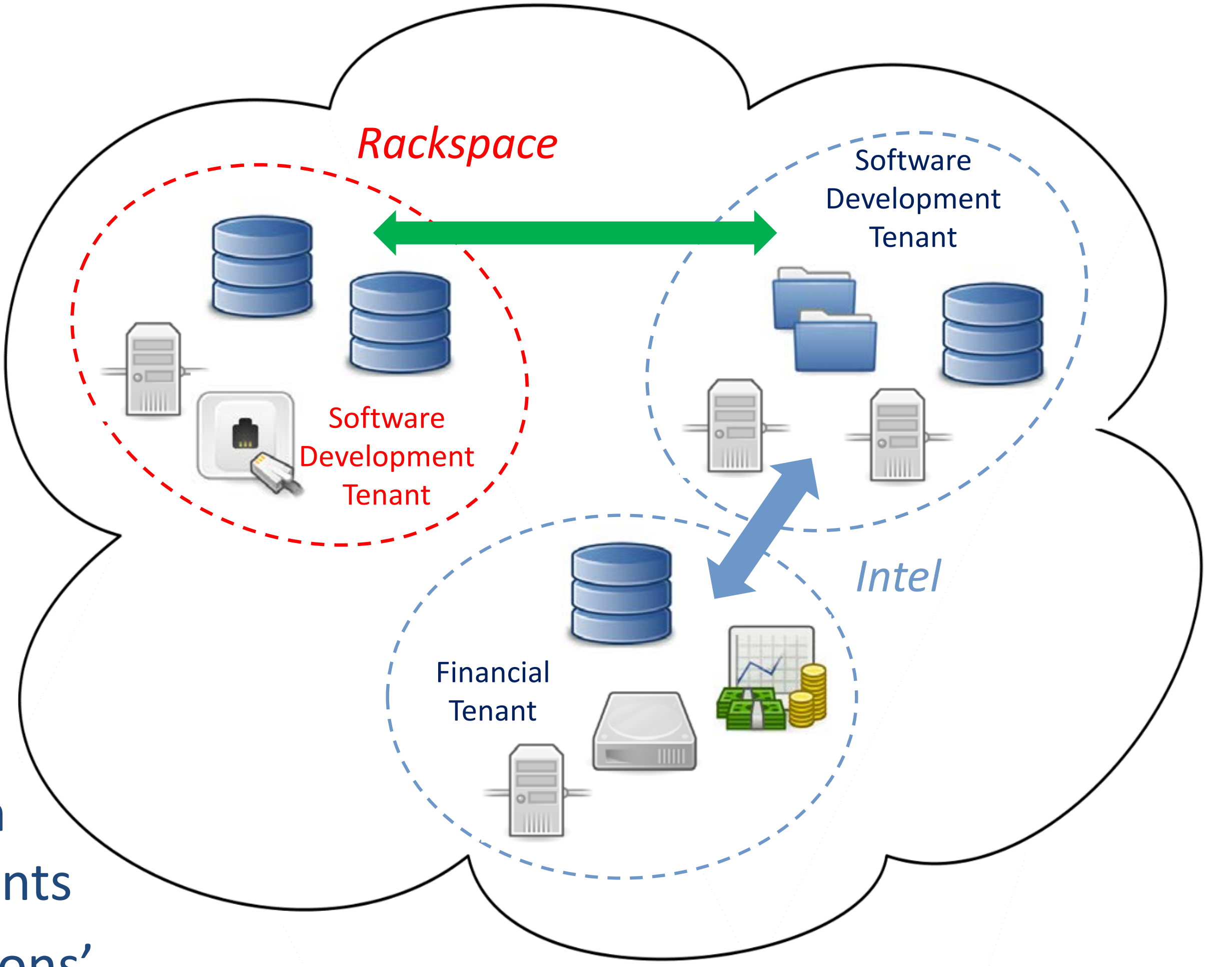
Navid Pustchi and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio

November 3-5, 2015
9th International Conference on Network and System Security

"Moving" to Cloud



Why Collaboration ?



Cloud Service Provider

- Large Organization with multiple tenants
- Distinct Organizations' Collaborative tasks

Scope of Contribution

- Contribution
  - An Attribute Based Access Control Model to enable collaboration between trusted tenants
  - Cross-tenant attribute assignment
  - Users cross-tenant access consistent with trust relation

- Scope
  - Infrastructure-as-a-Service (IaaS)
  - Single cloud
  - Multi-tenant

Multi-Tenancy

- Multi-tenancy
  - From Cloud Service Provider (CSP) perspective
    - Each customer bounded to a tenant, isolated from each other
    - Manages its own users and cloud resources
  - Tenant owner
    - An individual
    - An organization
    - A department of an organization

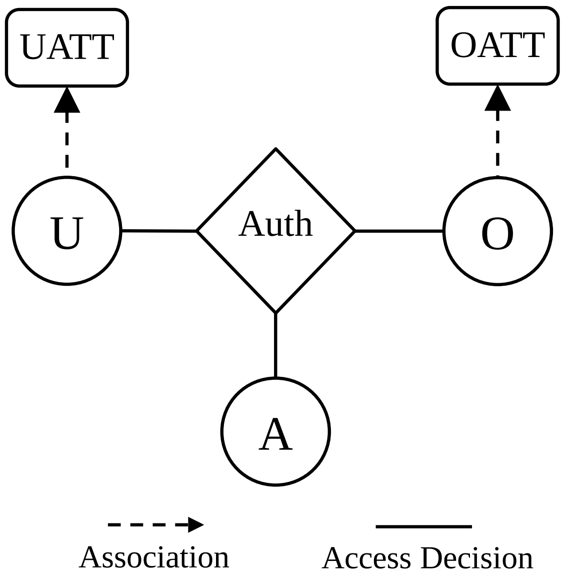
Attribute Based Access Control (ABAC)

- Attributes are *name:value* pairs
  - Represents user and resource properties
- Associated with
  - Users
  - Objects
  - Tenants
  - Contexts
- Converted to rights by authorization policies
  - In-time
  - Entity attributes
  - Set of actions

Why Another Model

- ABAC
  - RBAC shortcomings needs custom extension
    - For example real time environmental parameters.
  - ABAC is more flexible
    - Accommodate environmental parameters.
- MT-ABAC
  - Multi-tenancy
  - Collaboration consistent with trust

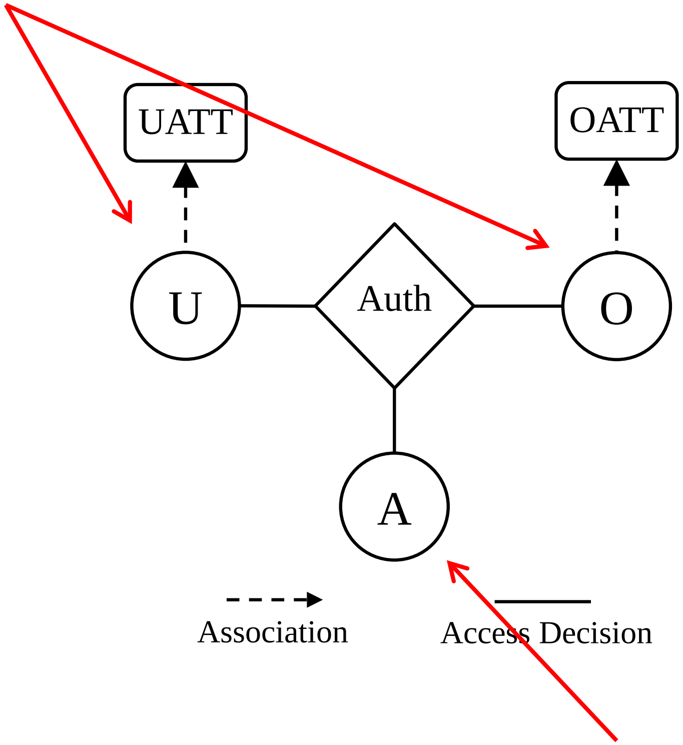
$ABAC_0$ Model Structure



World-Leading Research with Real-World Impact!

$ABAC_0$ Model Structure

Finite set of *users* and *objects*

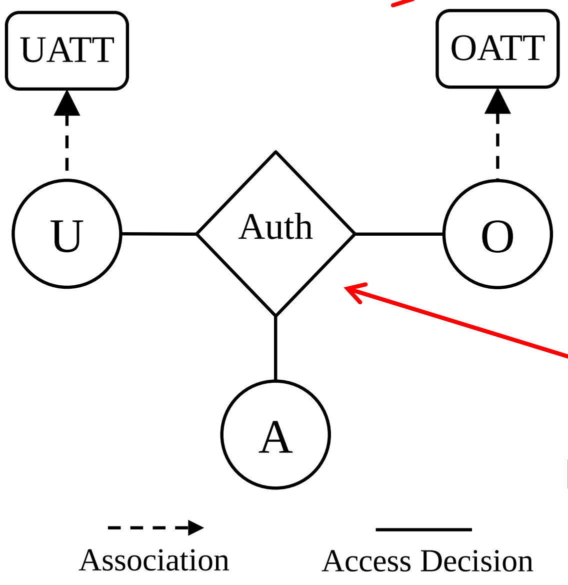


Set of *actions*

typically = {create, read, update, delete}

$ABAC_0$ Model Structure

Finite set of *user* and *object attribute functions*



A *user attribute* function Such as *Role* for a specific user $\boldsymbol{U_1}$ returns *cloud_admin*

Role($\boldsymbol{U_1}$) = *cloud_admin*

Policy Configuration point

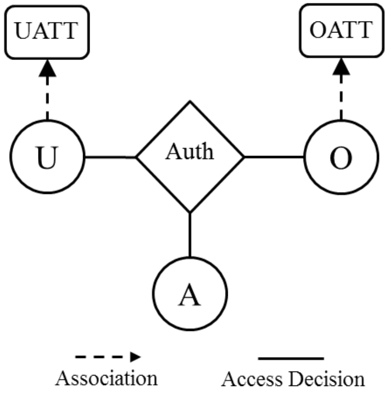
$ABAC_0$ Model Structure

➢ Attribute Functions

Each attribute function maps elements in U and O to atomic or set values as follows.

$$\forall uatt \in UATT.uatt : U \rightarrow \begin{cases} Scope(uatt)\ if\ attType(uatt) = atomic \\ 2^{Scope(uatt)}\ \ if\ attType(uatt) = set \end{cases}$$

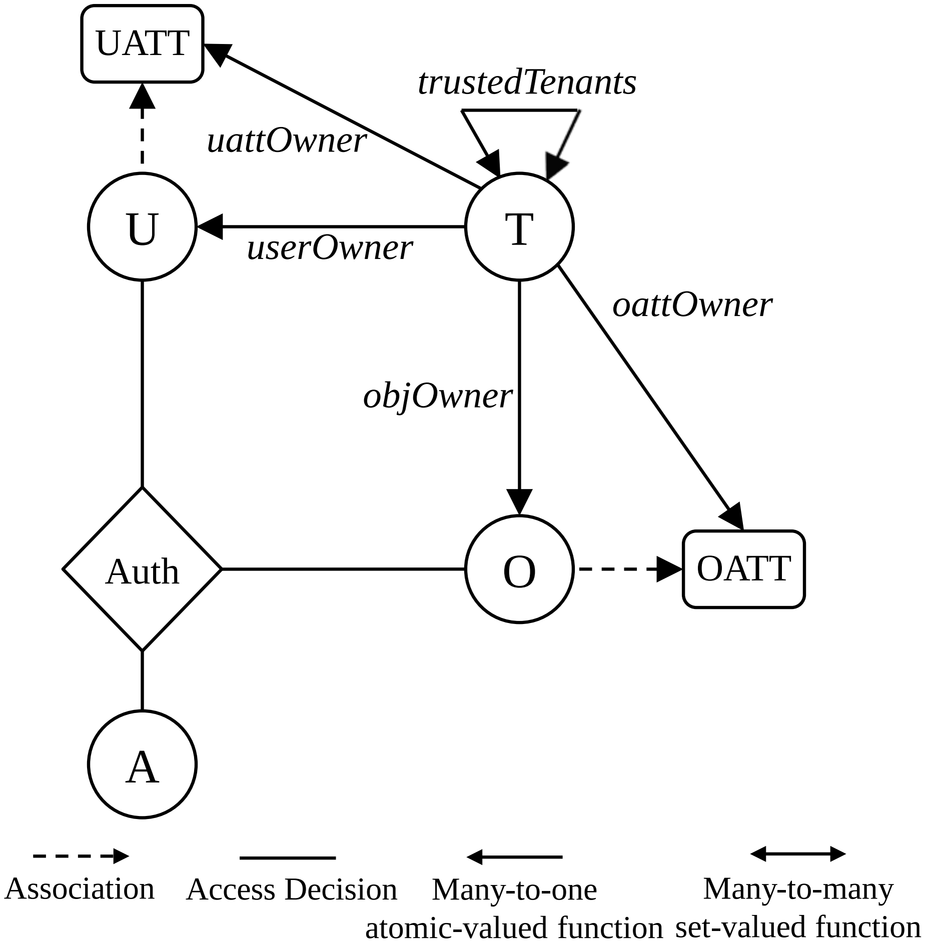
$$\forall oatt \in OATT.oatt : O \rightarrow \begin{cases} Scope(oatt)\ if\ attType(oatt) = atomic \\ 2^{Scope(oatt)}\ \ if\ attType(oatt) = set \end{cases}$$



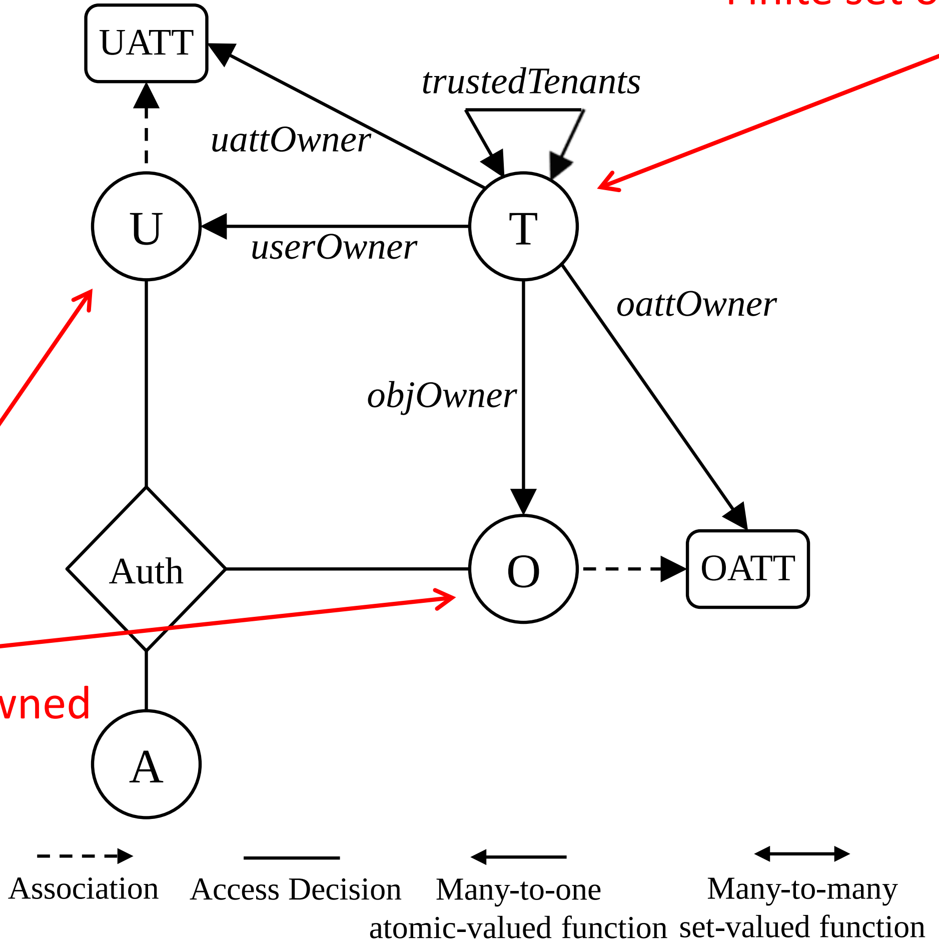
➢ Authorization Policy

$For\ each\ a \in A,\ Authorization_a(u : U, o : O)\ is\ a\ propositional\ logic\ predicate$

$MT-ABAC_0$ Model Structure



$MT-ABAC_0$ Model Structure

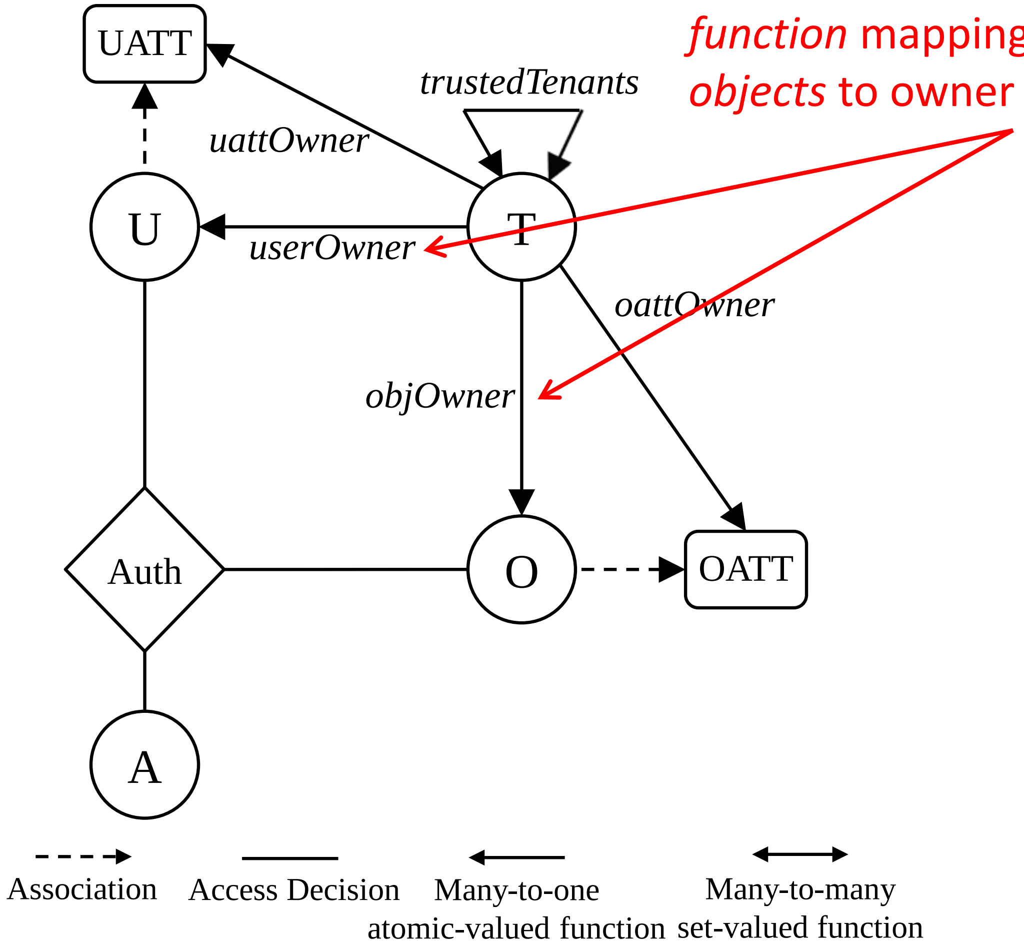


Finite set of existing *tenants*

Each *user* and *object* owned by a single *tenant*

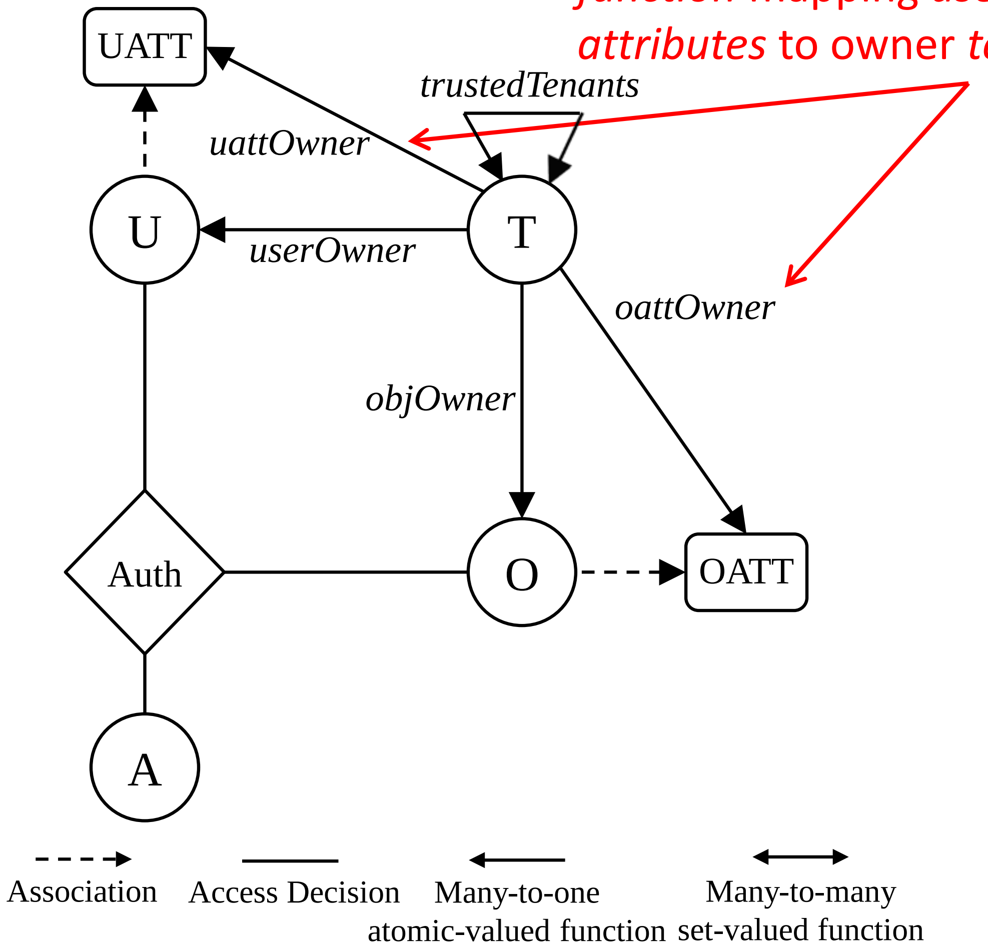
$MT - ABAC_0$ Model Structure

Required *atomic-valued attribute function* mapping *users* and *objects* to owner *tenant*

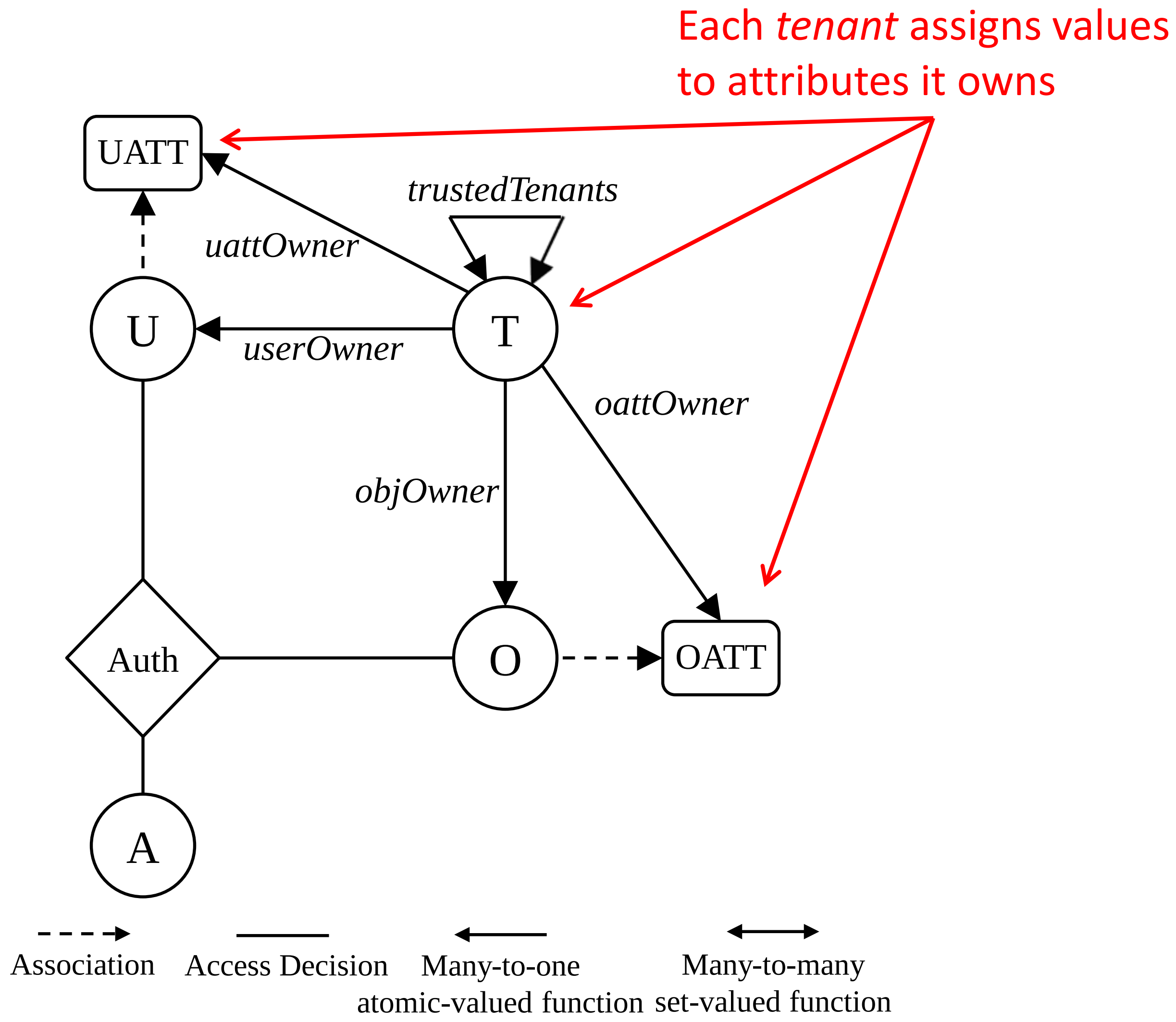


$MT - ABAC_0$ Model Structure

Required *atomic-valued meta-attribute function* mapping *user* and *object attributes* to owner *tenant*



$MT - ABAC_0$ Model Structure

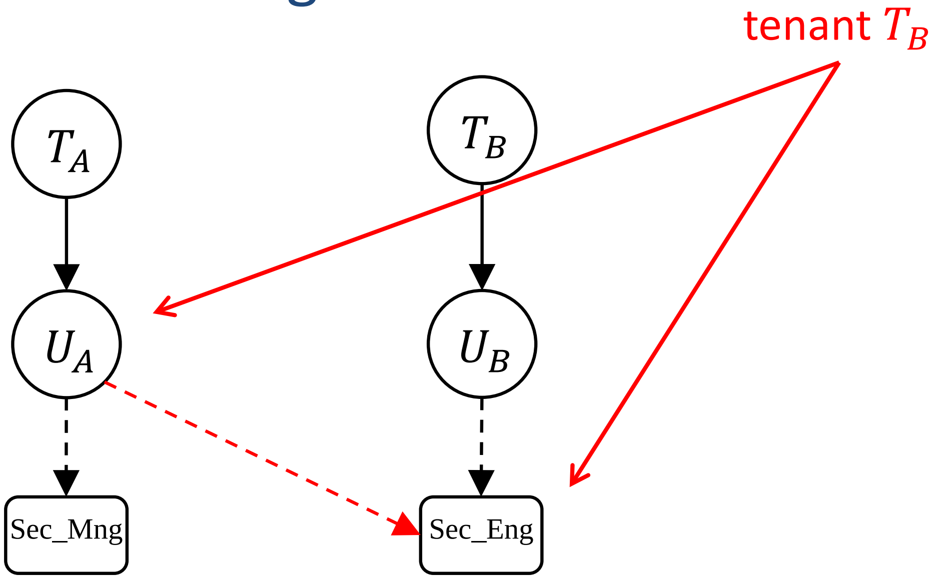


Each *tenant* assigns values to attributes it owns
UATT
trustedTenants
uattOwner
U
T
userOwner
oattOwner
objOwner
Auth
O
OATT
A
Association
Access Decision
Many-to-one atomic-valued function
Many-to-many set-valued function

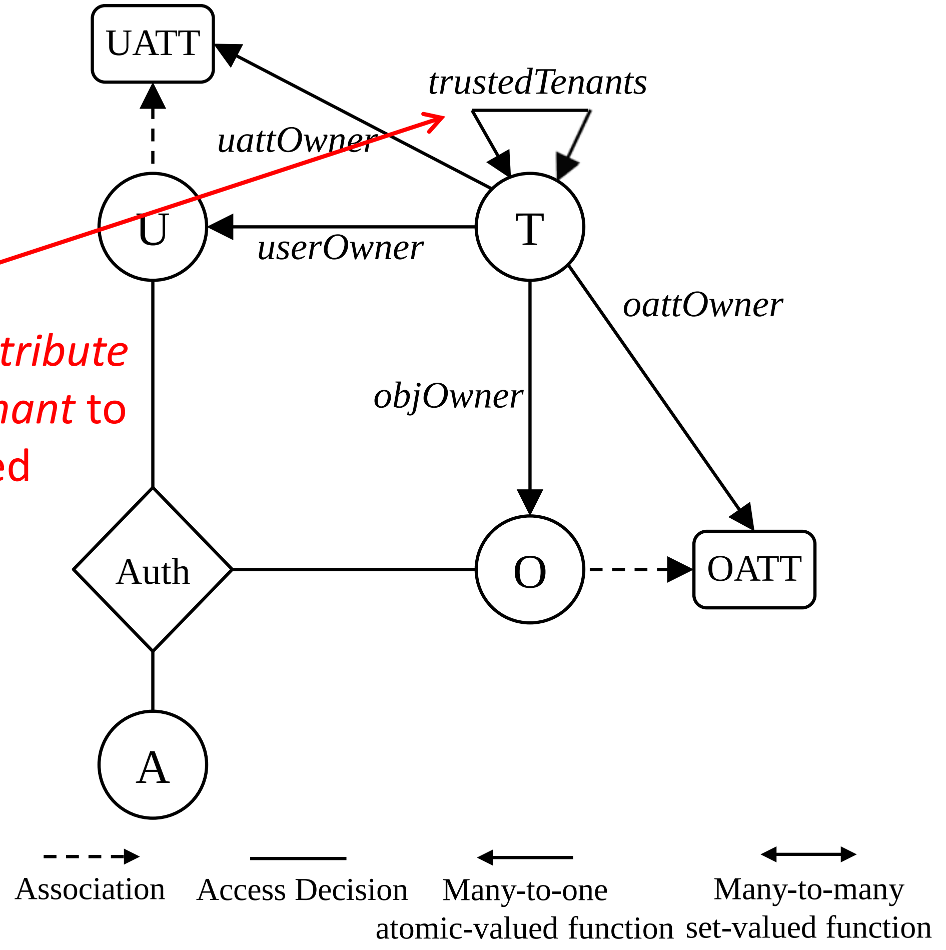
World-Leading Research with Real-World Impact!

Tenant-Trust

- Tenant-trust type-β
  - If $T_A \trianglelefteq_\beta T_B$, tenant T_B is authorized to assign values for T_B's user attributes to tenant T_A's users. Tenant $T_A$controls tenant-trust existence while T_B controls cross-tenant attribute assignments.

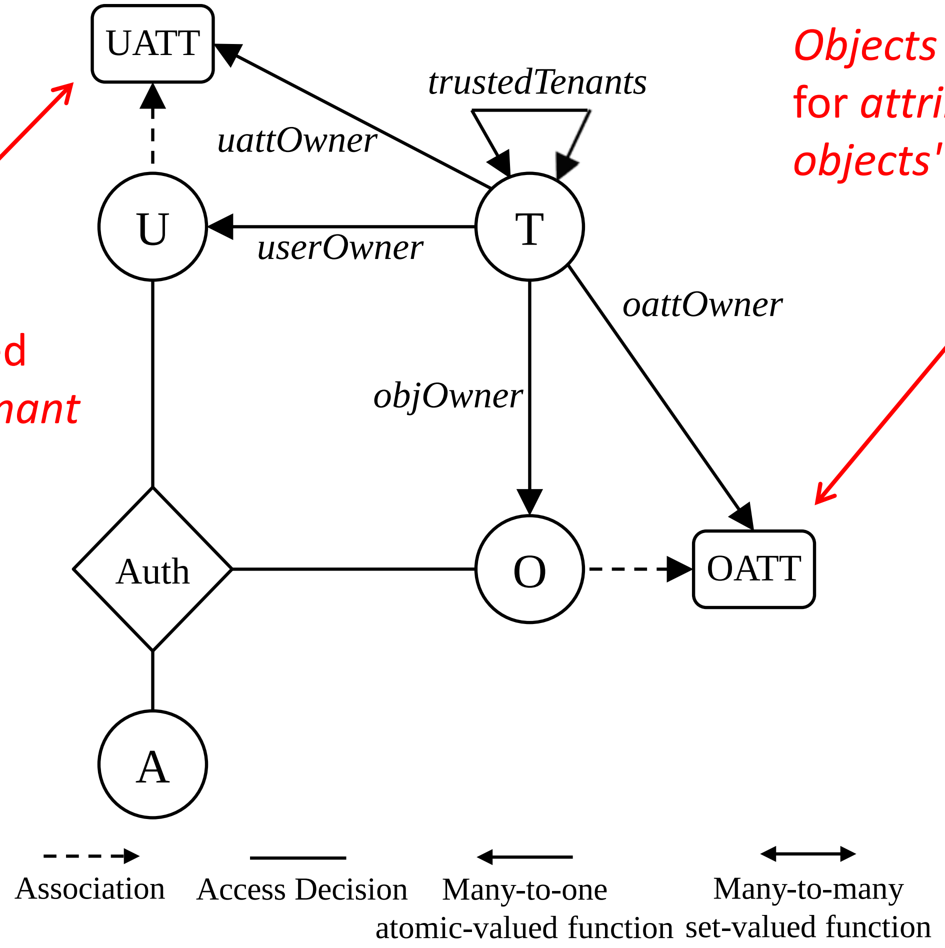


$MT - ABAC_0$ Model Structure



Required *set-valued attribute function* mapping *a tenant* to the power set of trusted *tenants*

$MT - ABAC_0$ Model Structure

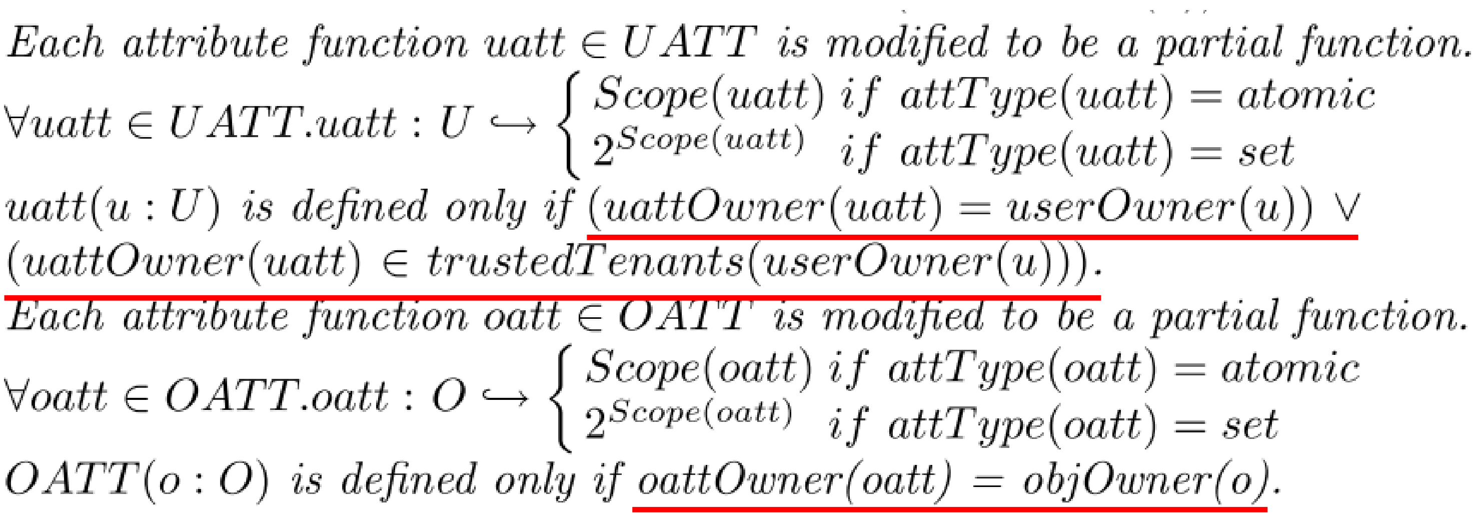


Objects only assigned values for *attributes* owned by *objects'* owner *tenant*

User attributes assigned values from owning *tenant* and trusted *tenants*

$MT - ABAC_0$ Model Structure

➢ Attribute Functions



Each attribute function $uatt \in UATT$ is modified to be a partial function.

$$\forall uatt \in UATT.uatt : U \hookrightarrow \begin{cases} Scope(uatt)\ if\ attType(uatt) = atomic \\ 2^{Scope(uatt)}\ \ if\ attType(uatt) = set \end{cases}$$

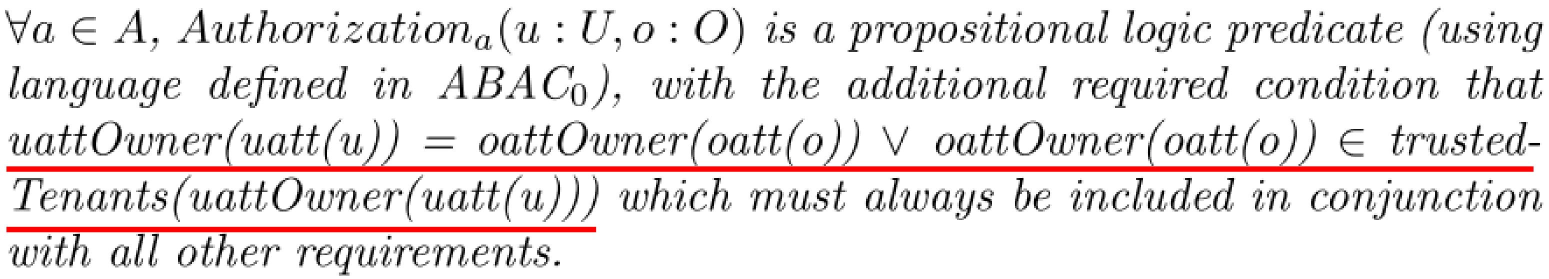
$uatt(u : U)$ is defined only if $\underline{(uattOwner(uatt) = userOwner(u)) \lor (uattOwner(uatt) \in trustedTenants(userOwner(u)))}$.

Each attribute function $oatt \in OATT$ is modified to be a partial function.

$$\forall oatt \in OATT.oatt : O \hookrightarrow \begin{cases} Scope(oatt)\ if\ attType(oatt) = atomic \\ 2^{Scope(oatt)}\ \ if\ attType(oatt) = set \end{cases}$$

$OATT(o : O)$ is defined only if $\underline{oattOwner(oatt) = objOwner(o)}$.

➢ Authorization Policy



$\forall a \in A$, $Authorization_a(u : U, o : O)$ is a propositional logic predicate (using language defined in $ABAC_0$), with the additional required condition that $\underline{uattOwner(uatt(u)) = oattOwner(oatt(o)) \lor oattOwner(oatt(o)) \in trustedTenants(uattOwner(uatt(u)))}$ which must always be included in conjunction with all other requirements.

$MT - ABAC_0$ Model Structure

➢ Attribute Functions

User attributes assigned values from owning *tenant* and trusted *tenants*

Each attribute function $uatt \in UATT$ *is modified to be a partial function.*

$$\forall uatt \in UATT.uatt : U \hookrightarrow \begin{cases} Scope(uatt) \ if \ attType(uatt) = atomic \\ 2^{Scope(uatt)} \quad if \ attType(uatt) = set \end{cases}$$

$uatt(u:U)$ *is defined only if* $\underline{(uattOwner(uatt) = userOwner(u)) \vee}$ $\underline{(uattOwner(uatt) \in trustedTenants(userOwner(u)))}$.

Each attribute function $oatt \in OATT$ *is modified to be a partial function.*

$$\forall oatt \in OATT.oatt : O \hookrightarrow \begin{cases} Scope(oatt) \ if \ attType(oatt) = atomic \\ 2^{Scope(oatt)} \quad if \ attType(oatt) = set \end{cases}$$

$OATT(o:O)$ *is defined only if* $\underline{oattOwner(oatt) = objOwner(o)}$.

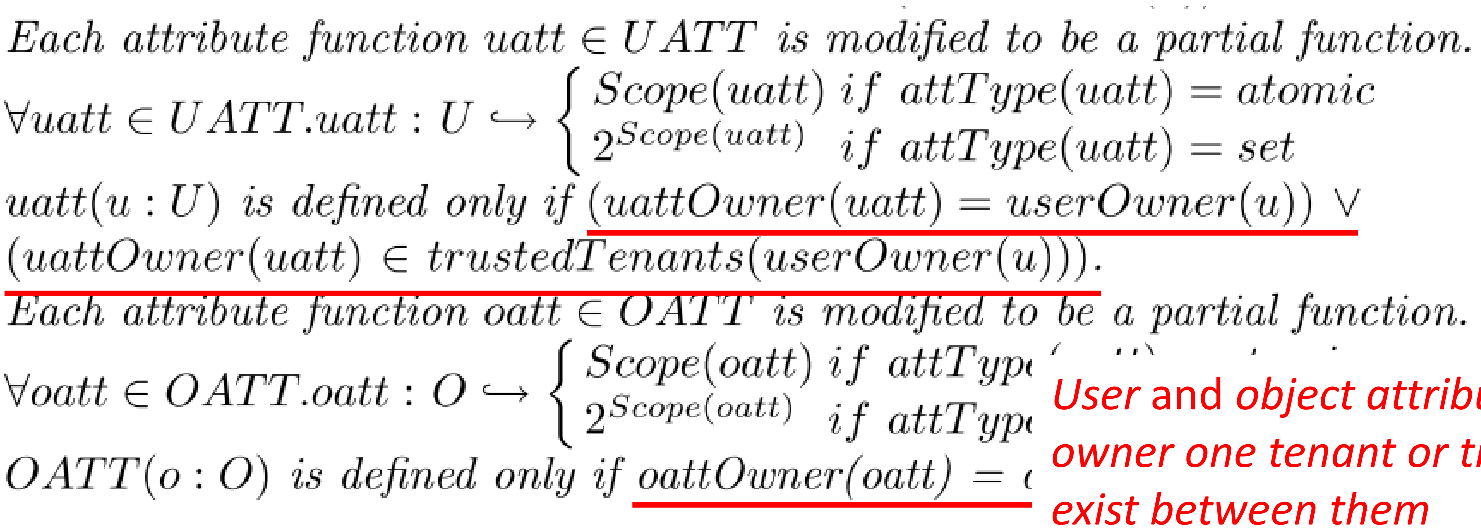
➢ Authorization Policy

$\forall a \in A$, $Authorization_a(u:U, o:O)$ *is a pro* … *ng language defined in* $ABAC_0$), *with the add* … *at* $\underline{uattOwner(uatt(u)) = oattOwner(oatt(o)) \vee}$ … $\underline{d\text{-}}$ $\underline{Tenants(uattOwner(uatt(u)))}$ *which must always be included in conjunction with all other requirements.*

Objects only assigned values for *attributes* owned by *objects'* owner *tenant*

$MT - ABAC_0$ Model Structure

- Attribute Functions



Each attribute function $uatt \in UATT$ *is modified to be a partial function.*

$$\forall uatt \in UATT.uatt : U \hookrightarrow \begin{cases} Scope(uatt)\ if\ attType(uatt) = atomic \\ 2^{Scope(uatt)}\quad if\ attType(uatt) = set \end{cases}$$

$uatt(u : U)$ *is defined only if* $\underline{(uattOwner(uatt) = userOwner(u)) \vee}$ $\underline{(uattOwner(uatt) \in trustedTenants(userOwner(u)))}.$

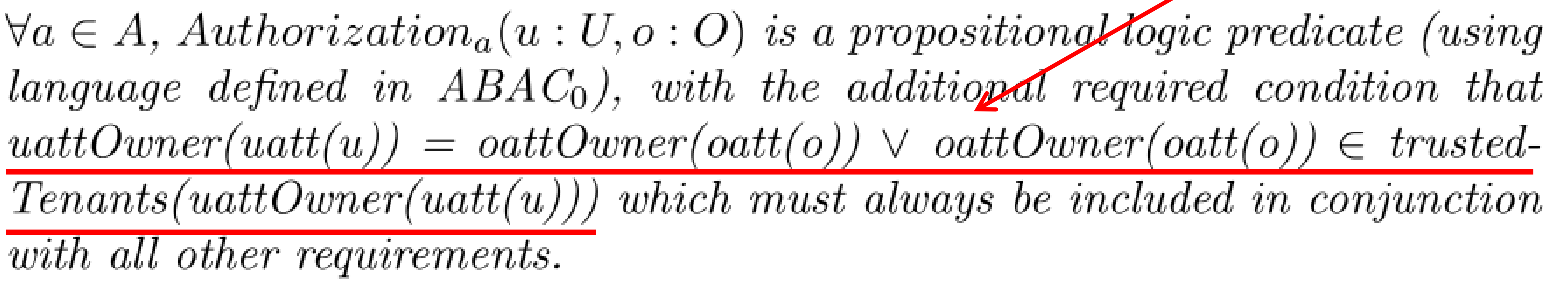
Each attribute function $oatt \in OATT$ *is modified to be a partial function.*

$$\forall oatt \in OATT.oatt : O \hookrightarrow \begin{cases} Scope(oatt)\ if\ attTyp \\ 2^{Scope(oatt)}\quad if\ attTyp \end{cases}$$

$OATT(o : O)$ *is defined only if* $\underline{oattOwner(oatt) =}$

User and object attribute owner one tenant or trust exist between them

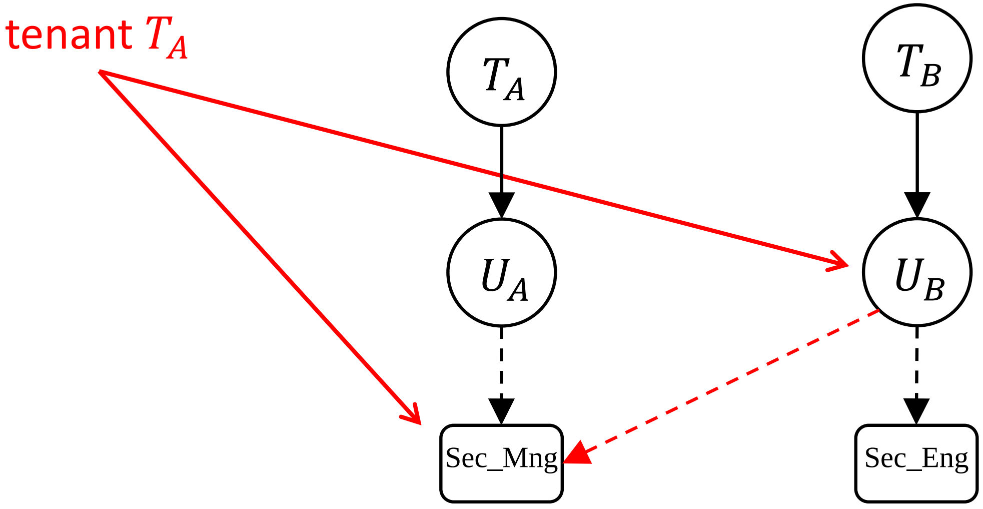
- Authorization Policy



$\forall a \in A$, $Authorization_a(u : U, o : O)$ *is a propositional logic predicate (using language defined in* $ABAC_0$*), with the additional required condition that* $\underline{uattOwner(uatt(u)) = oattOwner(oatt(o)) \vee oattOwner(oatt(o)) \in trusted\text{-}}$ $\underline{Tenants(uattOwner(uatt(u)))}$ *which must always be included in conjunction with all other requirements.*

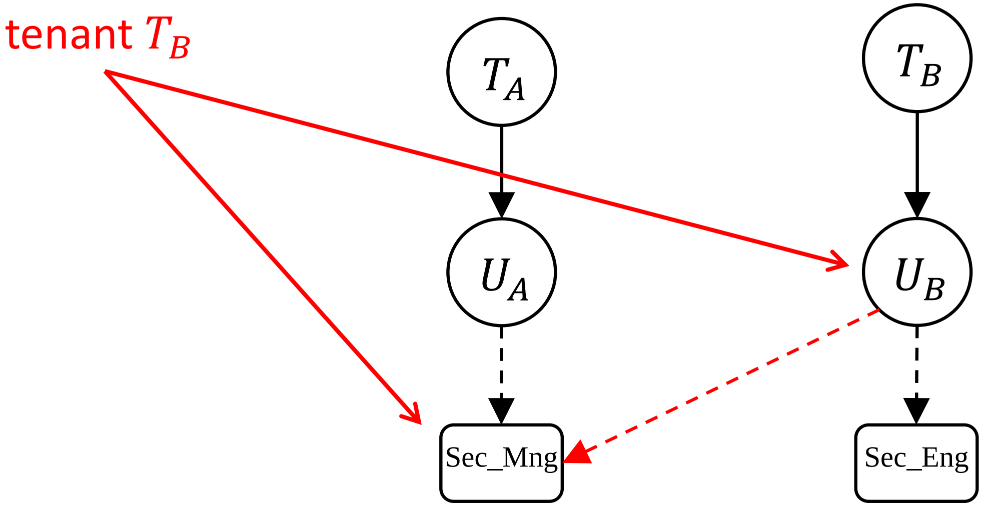
Tenant-Trust

- Tenant-trust type-α
  - If $T_A \trianglelefteq_\alpha T_B$, tenant T_A is authorized to assign values for T_A's user attributes to tenant T_B's users. Tenant $T_A$controls tenant-trust existence and cross-tenant attribute assignments.

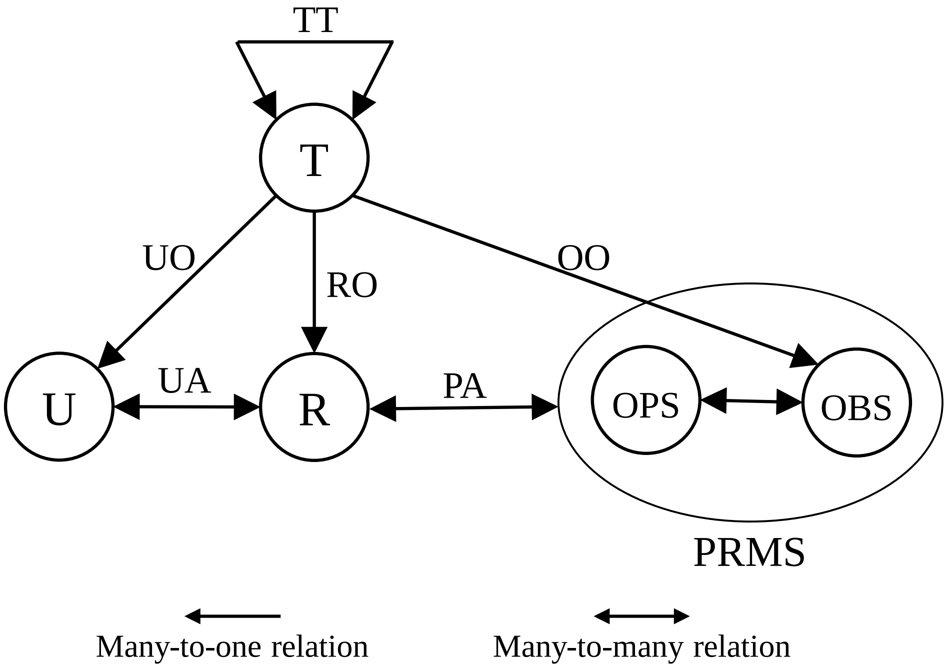


Tenant-Trust

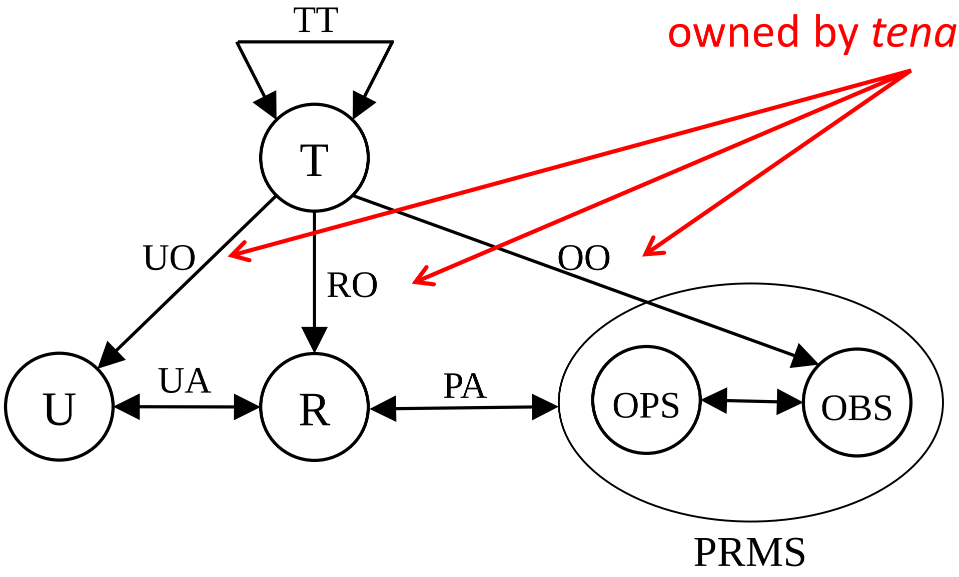
- Tenant-trust type-γ
  - If $T_A \trianglelefteq_\gamma T_B$, tenant T_B is authorized to assign values for T_A's user attributes to tenant T_B's users. Tenant $T_A$controls tenant-trust existence while T_B controls cross-tenant attribute assignments.



$MT-RBAC_0$ Model Structure



$MT - RBAC_0$ Model Structure



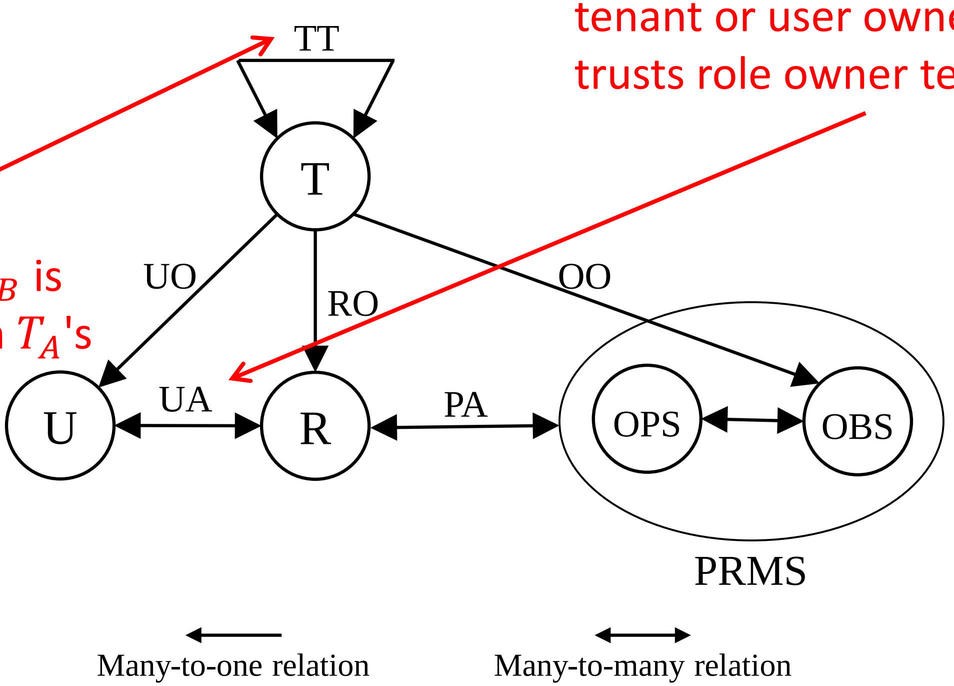
Users, *Roles*, and *Objects* owned by *tenants*

Many-to-one relation

Many-to-many relation

$MT-RBAC_0$ Model Structure

User-assignment is same tenant or user owner tenant trusts role owner tenant



If $T_A \trianglelefteq T_B$, tenant T_B is authorized to assign T_A's users to T_B's roles

Configuring $MT - RBAC_0$ to $MT - ABAC_0$

- Role as attribute
  - A set-valued attribute function $UserRole_j$ where *j* represents owner tenant.
  - A set-valued attribute function $ObjRole_{i,k}$ where *i* represents an operation and *k* owner tenant.
- Authorization

$$Authorization_i\ (u : U, o : O) = \bigvee_{k=1,\ldots,|T|} [userRole_k(u) \cap objRole_{i,k}(o) \neq \emptyset \wedge (t_k = userOwner(u) \vee t_k \in trustedTenants(userOwner(u))].$$

Conclusion

- Multi-tenant attribute-based access control model
  - Collaboration is enabled through cross-tenant attribute assignment.
  - Trust as a required attribute function.
  - Isolated attributes within tenants.

- Future Work
  - Other trust types.
  - Multi-cloud environments.
  - Relaxing object attribute assignments.