Computing (2014) 96:313–353
DOI 10.1007/s00607-013-0340-2

Enforcing spatio-temporal access control in mobile applications

Ramadan Abdunabi · Wuliang Sun · Indrakshi Ray

Received: 30 November 2012 / Accepted: 25 July 2013 / Published online: 21 August 2013

Abstract Mobile application technology is quickly evolving and being progressively utilized in the commercial and public sectors. Such applications make use of spatio-temporal information to provide better services and functionalities. Authorization to such services often depends on the credentials of the user and also on the location and time. Although researchers have proposed spatio-temporal access control models for such applications, not much has been done with respect to enforcement of spatio-temporal access control. Towards this end, we provide a practical framework that allows one to enforce spatio-temporal policies in mobile applications. Our policy enforcement mechanism illustrates the practical viability of spatio-temporal authorization models and discusses potential challenges with possible solutions. Specifically, we propose an architecture for enforcing spatio-temporal access control and demonstrate its feasibility by developing a prototype. We also provide a number of protocols for granting and revoking access and formally analyze these protocols using the Alloy constraint solver to provide assurance that our proposed approach is indeed secure.

Keywords Spatio-temporal access control · Policy enforcement

Mathematics Subject Classification 68N30

R. Abdunabi · W. Sun · I. Ray (✉)
Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA
e-mail: iray@cs.colostate.edu

R. Abdunabi
e-mail: rabdunab@cs.colostate.edu

W. Sun
e-mail: sunwl@cs.colostate.edu

1 Introduction

The growth of mobile device technology is spawning many new applications that make use of location information. One such example application is the use of global positioning systems (GPS) [47] to alert drivers for exceeding the speed limit in school zones [49]. A second example is in the area of mobile-healthcare that detects and alerts medical professionals of a patient's fall [13]. We can think of a situation in which a rogue user gets a hold of a military personnel's hand held device and use it in an unclassified environment to launch a missile. Although mobile applications can benefit various application domains including e-commerce, electronic government, healthcare, and power-control systems, security issues must be addressed before such systems can be widely deployed. Mobile devices may fall into the hands of attackers who may cause irreparable damage if adequate protection is not provided. Authentication and access control provide the first line of defense against any security attack. Existing access control mechanisms do not take user's mobility into account while providing authorization to resources. Thus, existing access control models cannot automatically disable access to a top secret resource when a user leaves the room. Spatio-temporal access control models are needed that determine access based on user's credential, location of the user and time of access.

Researchers have extended traditional role-based access control (RBAC) [41] to provide spatio-temporal access control [24,33,46]. In these models, access control depends on three factors: role of the user, the time and location of access. Most of the works on spatio-temporal RBAC associate two entities, namely, location and time with users, roles, and permissions. The location and time associated with a user correspond to his current time and his present location. The location and time associated with a role designate when and where the role can be activated. The location and time associated with a permission signify when and where a permission can be invoked. In addition, researchers have also suggested how spatio-temporal constraints can be associated with role hierarchy (RH) and separation of duties (SoD) constraints. Furthermore, a number of studies have formally analyzed spatio-temporal RBAC policies [3,34].

Very little research appears in addressing the challenges raised by enforcing spatio-temporal RBAC policies in practical applications. The enforcement mechanism needs to integrate the access control component into a typical application architecture. Access control models traditionally define policies from an abstract perspective and enforcement mechanisms describe how the policies are actually implemented in the context of the application [42]. The enforcement of spatio-temporal access control models in mobile applications introduces a number of interesting implementation challenges that have not been addressed successfully yet. Verifying the time and location of a mobile user while he is accessing some authorized resources is non-trivial.

The research work described in the paper focuses on addressing a primary problem of enforcing spatio-temporal access controls in a mobile environment. We demonstrate our ideas through a very simple spatio-temporal access control model for mobile applications. We introduce a platform-independent architecture for our spatio-temporal enforcement mechanism. Our model separates a security policy from the point of use, and thus makes it possible to be integrated in many applications. We develop a number of fully automated protocols that securely control the communication exchanges

for providing and maintaining access to application resources where the user may be on-the-move.

The protocols for requesting access or maintaining access may be subject to security breaches. Consequently, they must be analyzed to give assurance that security cannot be compromised through these protocols. We identify some common threats and discuss the safeguards that we take to protect against such threats. However, providing informal assurance for such complex protocols may not be adequate. Formal analysis must be done to provide assurance of security properties. Manual analysis is tedious and error-prone. Towards this end, we advocate the use of Alloy [8], which supports automated analysis, for protocol verification. Alloy is a modeling language capable of expressing complex structural constraints and behavior. It is supported by an automated constraint solver called Alloy analyzer that searches instances of the model to check for satisfaction of system properties. We demonstrate how the protocol can be modeled and the security properties verified in Alloy.

It is not enough to propose an architecture and a set of protocols for accessing resources. In order to demonstrate the feasibility of our approach, we develop a proof-of-concept prototype and show how spatio-temporal policies can be implemented for a mobile application. We describe the implementation details. We also conduct some experiments that give the delays incurred for performing authorization checks using our model.

The remainder of this paper is organized as follows. Section 2 enumerates some related work. Section 3 formally presents a simple spatio-temporal RBAC model. Section 4 provides the proposed architecture model. Section 5 introduces the resource access protocols. Section 6 discusses some potential attacks on our protocol. Section 7 formally analyzes the proposed protocols. Section 8 describes a prototype implementation and provides some performance results. Section 9 concludes the paper with pointers to future directions.

2 Related work

RBAC [41] is the de-facto access control model used in the commercial sector. RBAC is policy-neutral and can be used to express different types of policies [44]. In RBAC, users are assigned to roles and roles are associated with permissions. In a session, a user can activate a subset of roles assigned to her. The operations that a user can perform in a session depend on the roles activated by the user and the permissions associated with those roles. To simplify role management, roles are organized in the form of a hierarchy. A senior role may inherit the permissions of a junior role or a senior role may activate the junior role depending on whether the roles are connected by permission inheritance hierarchy or role activation hierarchy. In order to prevent fraud, RBAC allows one to specify separation of duty constraints. Static separation of duty constraints prevent a user from being assigned to conflicting roles or to a role being associated with conflicting permissions. Dynamic separation of duty constraints prevent a user from activating conflicting roles.

Researchers have worked on extending RBAC with time and location. Bertino et al. [12] proposed a temporal RBAC in which the concept of role enabling and disabling is introduced. Roles can be enabled or disabled on the basis of temporal constraints. Roles

can be activated if they are enabled. Joshi et al. proposed a generalized temporal RBAC (GTRBAC) [26] model that associates temporal constraints with the entities and the relationships in an RBAC model. Researchers have also extended RBAC using location information. Hansen and Oleshchuk [15] proposed the spatial RBAC (SRBAC) for specifying location based access control policies for wireless networks. Bertino et al. [11] proposed the GEO-RBAC model that allows role activation based on users' locations and permission-role assignment. Ray et al. [23] proposed a location aware RBAC model (LRBAC) that incorporates location constraints on user-role activation, user-role assignment, and permission-role assignment.

The use of both spatial and temporal information for managing access control has also been investigated by many researchers [3,24,29,33,45,46]. In these models, spatio-temporal constraints are applied to user-role assignment, permission-role assignment, and user-role activation. Some of these models also consider the impact of spatio-temporal constraints on role hierarchy and separation of duty. Various forms of role-hierarchy and separation of duty constraints have been proposed that are selectively enabled when spatio-temporal constraints are satisfied.

Researchers [3,34] have also investigated the use of Alloy for verifying spatio-temporal RBAC policies. In order to make the analysis tractable, Alloy requires the user to scope the problem. The results of the analysis are therefore applicable only for the scope of the problem being verified. Modeling and analyzing concurrency in Alloy is non-trivial. Towards this end, researchers [5,43] have investigated alternative techniques based on coloured Petri Nets and Timed Automata for verifying temporal, spatio-temporal and real-time RBAC policies. The major challenge in these works is how to perform the analysis without causing the state explosion problem. However, previous works do not address issues pertaining to enforcing spatio-temporal models in a mobile environment; this we do in our current paper.

Enforcement of spatio-temporal RBAC policies has received little attention. Kirkpatrick and Bertino [37] proposed protocols based on the GEO-RBAC model [11] which supports spatially-aware role-based access control policy. The authors assume that a number of location devices are incorporated in known physical spaces to provide location proof for users. The user presents the location proof together with other relevant credentials in the access request. First, we do not require location devices to be installed in given physical spaces and our approach provides more user mobility. Second, we support the temporary suspension of the user's access for a certain period of time when the user is moving out of a valid zone; this feature is not supported in the work of Kirkpatrick and Bertino. Third, for continuity of access, the authors work mandates a user to initiate a conformation protocol periodically to prove her location, even if she does not move. In our event-based approach, access termination takes place only when user moves out of the valid zone. We are thus able to reduce the communication overhead. Fourth, we provide a formal verification of our enforcement mechanism compared with the informal analysis of Kirkpatrick and Bertino. Fifth, we provide a more modular enforcement mechanism by separating a policy component from the point of use—this simplifies understanding, analysis, and evolution of the individual components.

In one of our previous works [40], we proposed a spatio-temporal role-based access control model for mobile applications and discuss a simple centralized archi-

tecture for enforcing policies adhering to this model. Our current work extends our earlier enforcement work in several ways. First, our proposed software architecture model is based on a distributed environment. Second, we discuss a number of issues (efficiency, generality) that guided our architectural design. Third, we provide an algorithm for generating an authorization token that is needed to access resources. Fourth, we develop a threat model to identify possible attacks on our proposed design and design countermeasures to tackle these attacks. Fifth, we provide a formal analysis approach to uncover vulnerabilities in the protocol design. We resort to the use of model finder Alloy that rigorously checks whether some attackers can break our protocol.

Enforcement has also been discussed in the context of XACML (eXtensible Access Control Markup Language) [31]. In a usage scenario of XACML, a subject sends a resource access request to the Policy enforcement point (PEP), which is the entity protecting that resource. The PEP creates a request based on the subject's credentials and other relevant information and sends it to the policy decision point (PDP). The PDP forms a request to the policy information point (PIP) for retrieving information relevant to that request. Once the PDP has determined the access decision, the decision is returned to the PEP, which then allows or denies access to the requester. We were inspired by the XACML principles, but deviated from some recommendations for reasons of efficiency.

Note that, existing XACML formulation of RBAC policies termed as XACML RBAC profiles (RB-XACML) [2,38] do not support the impact of spatio-temporal information on RBAC entities and relations. These RB-XACML profiles only consider the contextual information associated with subjects making resource access requests. Associating spatio-temporal information with RBAC components in RB-XACML profiles is a non-trivial task and there is no clear approach for formally verifying the modified RB-XACML profiles before deploying such policies.

3 Spatio-temporal access control model

Our spatio-temporal access control is based on RBAC [41]. In RBAC, users are assigned to roles and roles are associated with permissions. In a session, a user can activate a subset of roles assigned to her. The operations that a user can perform in a session depend on the roles activated by the user and the permissions associated with those roles. To simplify role management, roles are organized in the form of a hierarchy. A senior role may inherit the permissions of a junior role or a senior role may activate the junior role depending on whether the roles are connected by permission inheritance hierarchy or role activation hierarchy. In order to prevent fraud, RBAC allows one to specify separation of duty constraints. Static separation of duty constraints prevent a user from being assigned to conflicting roles or to a role being associated with conflicting permissions. Dynamic separation of duty constraints prevent a user from activating conflicting roles.

Figure 1 shows our spatio-temporal RBAC model (STRBAC) which is adapted from our earlier proposed model [40]. Since the model is not central to this paper, we only present those aspects that are needed to understand this work. The entire model specified in the unified modeling language (UML) with object constraint language (OCL) constraints appears in Appendix C. The model allows one to expresses policies

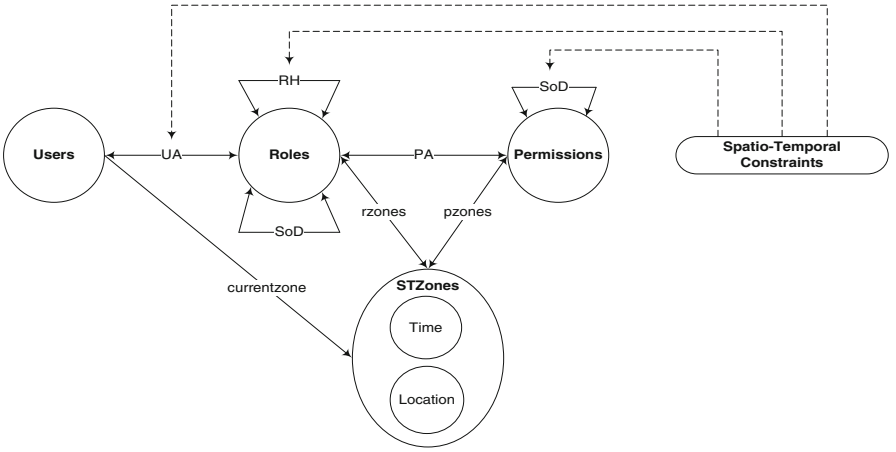

Fig. 1 A spatio-temporal RBAC model

where access depends on the role and location of the user and time of access. We assume that a location device is attached to a user that transmits his spatial coordinates at any given point of time. These devices are tamper-proof, so that malicious programs cannot spoof location or time information. The correct behavior of an application is ensured by the satisfaction of model predicates.

The essence of STRBAC is to permit access only when a user is in designated locations at specified times; when the user moves or the time of access expires then the access is revoked. Although our original model [40] supports spatio-temporal constraints on user-role assignment, permission-role assignment, user-role activation, role hierarchy, and separation of duties, we do not consider role hierarchy in this paper. Note that, adding role hierarchy will not add anything new from the research perspective, but it will make the presentation more complex. At the heart of our model is the concept of *spatio-temporal zone* that is a logical and abstract entity encapsulating location and time. The spatio-temporal zone z_i is represented as a pair: $< l_i, d_i >$ where l_i represents a spatial component and d_i represents a temporal one.

The set of spatio-temporal elements that are of interest to a mobile application is denoted as *STZones*. Model entities are associated with *STZones* in order to define where and when these entities are accessible. It defines a number of functions that associate each model entity with a subset of *STZones*. The *rzones* function ($rzones : Roles \rightarrow 2^{STZones}$) determines a unique subset of role zones that signifies the spatio-temporal coordinates where a given role can be assigned or activated. For example, the role of on-campus students can be activated or assigned only when the student is on-campus during a given semester. The function *pzones* ($pzones : Permissions \rightarrow 2^{STZones}$) defines the permission zones which are the set of spatio-temporal coordinates where the permission can be invoked. For example, the system administrator is allowed to perform backups at 10 pm on Fridays at the lab. The *currentzone* ($currentzone : Users \rightarrow STZones$) identifies the current spatio-temporal zone of a user at the access time. The model defines the following relationships in the context of *STZone*:

Role-assignment (UA): In order to activate a role, the user must be assigned to the role. For example, a student can activate on-campus role during a semester only if he is assigned the role of an on-campus student. This spatio-temporal assignment constraint is formalized in this model using an $assignRole(u, r, z)$ predicate. The predicate $assignRole(u, r, z)$ holds when the current user zone z is one of the zones in which role r is available. The predicate returns true if user u can be assigned role r in zone z, if z belongs to the zone in which r can be assigned.

- $assignRole(u, r, z) \Rightarrow z = currentzone(u) \wedge z \in rzones(r)$

Permission-assignment (PA): The predicate $assignPerm(r, p, z)$ expresses the spatio-temporal constraints on permissions assignment. This predicate is true if zone z is one of the zones where permission p can be invoked and role r is available.

- $assignPerm(r, p, z) \Rightarrow z \in (rzones(r) \cap pzones(p))$

Role-activation: In order for a user to access some resource, he needs to activate some role that gives her the permission to access it. Note that, a user can only activate those roles that have been assigned to her. A person to enter conference reviews must activate a program committee member role. She can do this only if she has been assigned the role of program committee member. In order for a user to activate a role, the predicate $activateRole(u, r, z)$ should be satisfied. This predicate holds if the current user zone is z and role r is available to user u in zone z.

- $activateRole(u, r, z) \Rightarrow z = currentzone(u) \wedge r \in assigned\_roles(u, z)$, where $assigned\_roles(u, z)$ returns the assigned roles to user u in zone z.

STRBAC model also specifies separation of duties (SoD) principle between roles and permissions to prevent fraud. SoD constraints are applied to conflicting roles or permissions in some spatio-temporal zones. We look at two such relations $RSoD$ and $PSoD$ in this paper. $RSoD$ prevents the same user from activating conflicting roles in certain spatio-temporal zone. For example, the same person cannot activate both the student role and the teaching assistant role during a lab session. $PSoD$ prevents the same role from invoking conflicting permissions in certain spatio-temporal zones. For example, the same role cannot be given the permission of chairing the session and presenting the paper in the same session in a conference. These relations are formally defined as follows:

- $RSoD \subseteq Roles \times Roles \times STZones$
- $PSoD \subseteq Permissions \times Permissions \times STZones$

Role SoD: A pair of mutual exclusive roles related by $RSoD$ are not allowed to be activated by the same individual in some spatio-temporal zones. The constraint is given below:

- $\forall u \in Users \bullet (r, r', z) \in RSoD \Rightarrow \neg(r \in active\_roles(u, z) \wedge r' \in active\_roles(u, z))$, where $active\_roles(u, z)$ gives the current roles activated by user u in zone z.

Permission SoD: Mutual exclusive permissions defined by $PSoD$ cannot be assigned to the same role in some zones. This requirement is expressed is follows:

- $\forall r \in Roles \bullet (p, p', z) \in PSoD \Rightarrow \neg(p \in assigned\_perms(r, z) \wedge p' \in assigned\_perms(r, z))$, where $assigned\_perms(r, z)$ defines the set of permissions associated with role r in zone z.

4 Software architecture model

In this section we describe a platform-independent implementation architecture of our model, which maps the high-level policy definition to the enforcement mechanism in mobile applications. The design is developed with aim of having a feasible trade off between security, functionality, and ease-of-use. In keeping with the principles of the usage control model $UCON_{ABC}$ [25], we do both pre-authorization and ongoing authorizations. Pre-authorization in $UCON_{ABC}$ requires that the resource access verification must be performed before the requested resource is accessed. Ongoing authorizations in $UCON_{ABC}$ requires that a system continue to enforce access control while the resource is being accessed. In spatio-temporal access control, both of these require the authenticity of the location and temporal information associated with the access. Since users are on the move in a mobile environment, this imposes additional challenges.

4.1 Architectural considerations

We first present some of the issues that we considered before designing our architecture.

- We decoupled the policy components from the application functionality components of the architecture. This decoupled approach allows one to change one component without affecting the other so long as the interface is not changed.
- We wanted to have a distributed architecture for reasons of performance and minimizing single point of failure. Specifically, different servers are involved in processing various aspects of the access control requests. Use of multiple servers and division of tasks among them help in improving the performance. In the subsequent sections, we describe the components of the architecture and demonstrate how they are distributed.
- We wanted to minimize the involvement of a trusted third party.
- We do not place any restrictions on the security of the communication channels.
- Users in our model are not confined to any specific locations.
- We tried to minimize computation and communication costs to the extent possible. For instance, policy enforcement point and policy decision point are done at the same server.
- We focused on minimizing the computations and communications performed on the client side to accommodate the resource constraints associated with the mobile devices. Major computations are performed on the server side.

4.2 Assumptions

Here we discuss our assumptions.

- Modules enforcing spatio-temporal access control are implemented by tamper-proof and trusted software.
- Identical public key cryptography and hash algorithms *MD5* should be installed at the client and server sides.

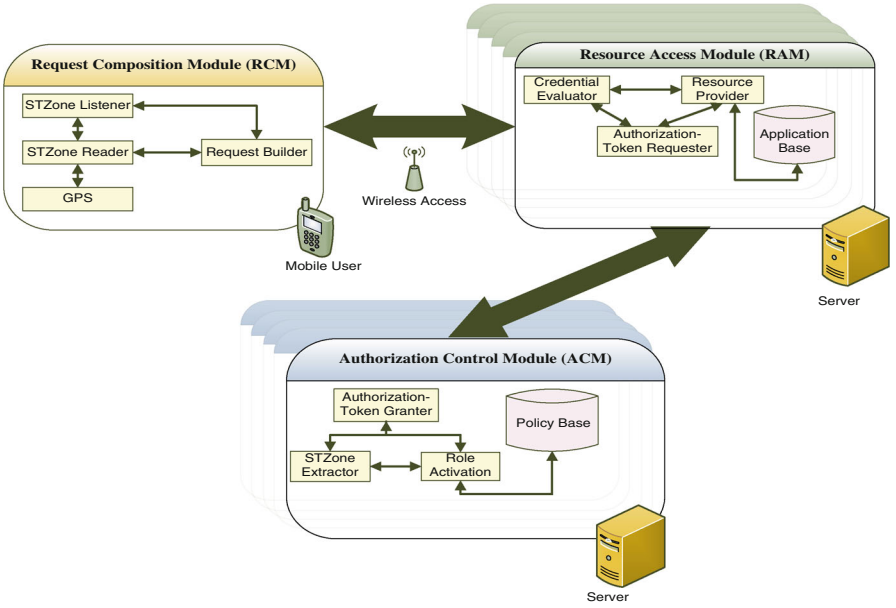

Fig. 2 Architecture of the spatio-temporal enforcement mechanism

- The strength of public key cryptography, hash algorithms, and password are chosen such that they cannot be compromised taking into account the resources of the attacker.
- In public key cryptosystem, a public and private key pair is associated with an entity. In our case, the entity consists of the user-device pair, given by *(user id, device id)*. Public key certificates are issued for each such pairs by the certificate authority. This ensures that a user cannot use his colleague's cell-phone to access an application. The certificate is issued by an authority trusted by all the entities.
- A tamper-proof location system such as GPS or WLAN-based systems [47] is installed in a user mobile device. Note that, smartphones with tamper-proof GPS receivers [18] onboard are becoming increasingly common. Most of today's smartphones, such as the iPhone/iOS, Bada, Windows Mobile 7, and Android, are fully GPS-enabled.
- The clocks present in the various modules are synchronized.
- The data consistency at all architecture components is guaranteed by applying some underlying protocols for consistent data services [6,9,32]. These protocols support the data integrity, concurrency control, and recovery from crashes and communication failures across interconnected databases.

4.3 Architecture modules

Figure 2 depicts the implementation architecture for enforcing spatio-temporal policies in a mobile application. The architecture in Fig. 2, consists of three core components: *request composition module (RCM)*, *resource access module (RAM)*, and *authorization*

control module (ACM) which are stand alone programs. *RCM* is installed in the user's mobile device while the *RAM* and *ACM* are implemented in servers which are not required to be co-located. The application data and policy data can be kept in distributed servers. In case the relevant data is not stored locally, the module must communicate with the remote servers.

We describe the proposed architecture modules in more details.

- *RCM* is responsible for forming a user access request and maintaining the access while the rights are being exercised. The *Request Builder* component in *RCM* creates a resource access request package that includes the current user *STZone* and user's access credentials (user identifier, password, timestamp, and device identifier) with the request and sends it to one of the available *RAM* servers. The *Request Builder* gets the current user's *STZone* from the *STZone Reader* which has access to the GPS location information and mobile device time information. *STZone Reader* tracks current user's *STZone* and sends notifications of invalid *STZone* to the *STZone Listener*. On receiving invalid zone information, *STZone Listener* sends an access termination request to the server.
- *RAM* is an intermediate server between a user's mobile and the *ACM* server. It receives the user's request and consults with the *ACM* to decide whether the user will get access to the resource. The *Credential Evaluator* component checks user credentials stored in the *Application Base*. The *Authorization-Token Requester* component requests an *authorization-token* from one of the available *ACM* servers. The *Resource Provider* component provides access to some object requested by users. It acts as the policy enforcement point (PEP) as it is responsible for handling the application resources accessed by the users.
- *ACM* is responsible for the policy evaluation and issuing a new *authorization-token* for every user request. *ACM* communicates with the *Policy Base* in order to issue users' *authorization-tokens*. The *Role Activation* maps a user to an appropriate set of roles and permissions based on its *STZone* provided by the *STZone Extractor* component. Furthermore, the *Role Activation* updates the policy state for each activated role. The *Authorization-Token Granter* is responsible for granting the *authorization-token* for a given role. An *authorization-token* can be granted only if the given role can be activated. Note that, a user's role can be activated if all the following conditions are satisfied: (a) the role is not already active, (b) the role can be activated in the given *STZone*, and (c) no conflicting roles are active. *ACM* acts as the Policy Decision Point (PDP) because it makes access decisions. It also acts as the Policy Implementation Point (PIP) because it issues *authorization-tokens* that identifies which resources can be accessed.

The *authorization-tokens* are tickets that allow client applications access to resources. An *authorization-token* is denoted as $AT = (ID_u, ID_{ut}, r, P, STZone)$ where ID_u refers to a user identifier, ID_{ut} is a token's identifier, r is the activated role, P is the set of permissions associated with role r, and *STZone* defines where and when these privileges are valid.

ACM grants or denies *authorization-tokens* using Algorithm 1. The algorithm takes in three inputs: user u, role r, and *STZone* z: u is the user who wants to activate role r in zone z. Line 1 initializes the permission P to the null set. In Lines 2 and 3 we check

if r has already been activated by u in zone z or u is not assigned to role r in zone z. If these conditions are true, then authorization tokens are not issued and a message is sent to the user in Line 32. If the user already has authorization tokens, a check is made to ensure that no conflicting roles or permissions are activated—this is done in Lines 5—25. If these conditions are satisfied, the role r can be activated for user u in zone z. The set of active roles for user u in zone z is updated in Line 26 to include the role r. In Line 27, P is set to the permissions associated with role r in zone z. Lines 28 and 29 generate the *authorization-token* AT. The algorithm produces as output the AT if the role can be activated, otherwise a reject message is sent to the user.

```
   input : User u, STZone z, and Role r
   output: authorization-token AT or Reject
1  P ⟵ ϕ;
2  if r ∉ active_roles(u, z) // r is already activated by u in zone z
3  and r ∈ assigned_roles(u, z) // r is assigned to u in zone z
4  then
5      if userToken(u) ≠ ϕ // u has some authorization-token
6      then
7          foreach r' ∈ active_roles(u, z) // checks conflicting roles
8          do
9              if r' ∈ RSoD(r, r', z) // a role conflict is exist
10             then
11                 print(Request is Rejected + Details);
12                 exit;
13             end
14         end
15         foreach p ∈ assigned_perms(r, z) // checks conflicting permissions
16         do
17             foreach p' ∈ assigned_perms(r, z) do
18                 if p' ∈ PSoD(p, p', z) // a permission conflict exists
19                 then
20                     print(Request is Rejected + Details);
21                     exit;
22                 end
23             end
24         end
25     end
26     active_roles(u, z) ⟵ active_roles(u, z) ∪ {r}; // updates role r state
27     P ⟵ assigned_perms(r, z);
28     IDut = generateID(AT); // generate a sequence number of user's token
29     AT ⟵ (u, IDut, r, P, z); // creates authorization-token AT
30 end
31 else
32     print(Request is Rejected + Details);
33 end
```

Algorithm 1: Generate *authorization-token* AT

With distributed and replicated *RAM* and *ACM* servers, *ACM* broadcasts the new *authorization-token* to all the *RAM* and *ACM* servers to keep the access control consistent.

4.4 Computational capabilities and storage space

The user's mobile device has less processing and storage capacity compared to that of the servers. However, today's mobile devices have adequate processing and storage capacity to execute our protocols. Our enforcement mechanism assumes that users' mobile devices can execute public key cryptography algorithms as well as hashing algorithms, such as *MD5*. Users' mobile devices must be able to store their own private key PrK_u, the *RAM*'s public key, and a list of *authorization-tokens*. Such data should be stored in a secure component of the user's smartphone; it should only be accessible by licensed applications.

RAM servers should also have the processing capability needed to do public key cryptography and hashing. A *RAM* server maintains a list of users' public key certificates denoted by $UCerts$. It also maintains a list of *authorization-tokens* that is referred to as the *UATokens* list. When an *authorization-token* expires, it must be removed from *UATokens* list. Since the enforcement mechanism may involve a distributed environment, *RAM* also stores a list of public keys certificates of the *ACM* servers referred to by the $ACCert$ list; these are needed to securely communicate with the various *ACM* servers. Each *ACM* server only stores its private key and a list of public keys certificates for distributed *RAM* servers. Note that, all such sensitive information must be stored in a secure and tamper-proof component.

5 Resource access protocols

We define a number of protocols for ensuring that resources are accessed with the correct authorization. We start with discussing how the initial access is granted and how the access is revoked when the environment changes.

5.1 Protocols prelude

The client and the servers must agree on the public key cryptography and hash algorithms used in the protocol. The clocks of the various entities must be synchronized before the start of the protocol.

We expect the user to register with the system and to create a password before the execution of the protocol. When a user registers in a system, a trusted *Certificate Authority* (*CA*) issues a pair of public and private keys for that user. A user's certificate has a user's public key PuK_u, a user's smartphone identifier ID_s, user identifier ID_u, and user password P_u. Prior to the start of the protocol, we assume that the *RAM*s', *ACM*s' public key certificates have already been distributed by the *CA* to *RCM*s, *RAM*s, and *ACM*s.

Figure 3 describes the communication exchanges of the resource access protocols implemented by a single *RCM* client, *RAM* server, and *ACM* server. Suffixes associ-

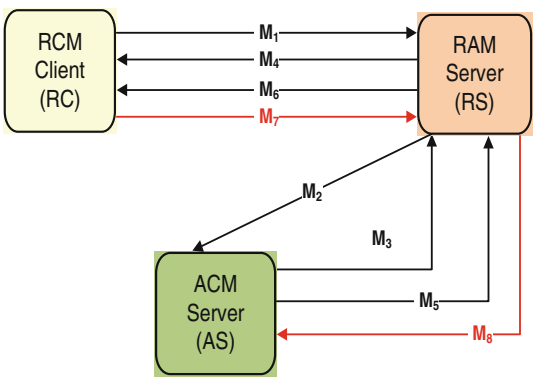

Fig. 3 Communication steps of the resource access protocols

Table 1 The notations of the resource access protocols

Symbols	Interpretation
ID_x	Identifier of party x
ID_s	User's device identifier
PuK_x	Public key of party x
PrK_x	Private key of party x
Ts_x	Timestamp computed by party x
P_u	User password
P_u^*	One-time password
$H(P_u, Ts_x)$	Computes P_u^*
M_x	Package payload created by party x
$E_x\{S\}$	Encryption of sequence S by PuK_x
M_x^*	Checksum on message M generated by party x
$H(M_x, Ts_x)$	Computes M_x^*
$S_x\{M_x^*\}$	Signing M_x^* by PrK_x
$A \xrightarrow{M_i} B$	Party A sends package M_i to party B
Tw	Time window
AT	Authorization-token
ID_{ut}	Authentication-token identifier
"Close"	Keyword indicates deletes user's AT
"Freeze"	Keyword indicates suspends user's AT

ated with the communication messages indicate the order of steps in the protocols. Message $\mathbf{M}_i$ indicates step i of the protocol. Table 1 enumerates the notations used in the description of the protocols. The *RAM* and *ACM* servers are implemented as two distinct centralized servers. The server hosting the *RAM* also houses the *Application Base* and is referred to as the *Resource Server* (*RS*). *ACM* along with the *Policy Base* are housed in a server called the *Authorization Server* (*AS*). *RCM* is implemented in a mobile device referred to by (*RC*). The channel between user mobile phone and *RAM* server is connected through a wireless network (GPRS, UMTS, WLAN, Internet WAP 2.0). We do not put any assumptions about the communication channels between the *RAM* servers and *ACM* servers: they can be either wired or wireless.

5.2 Initial access protocol

The steps of the basic resource access protocol for handling users' requests are performed as following:

– *Resource usage request* $[RCM \xrightarrow{M_1} RAM]$: a user's mobile sends an encrypted and digitally signed access request package M_1 to *RAM* as shown in Fig. 3. *RCM* creates an access request payload $M_{rc} = (ID_u, ID_s, P_u^*, STZone, r, Ts_{rc})$, where ID_u is the user identifier, ID_s is the device identifier, $P_u^* = H(P_u, Ts_{rc})$ is the user one-time password, $STZone$ is the current user zone, r is the requested

role, and Ts_{rc} is the timestamp at which M_{rc} is created. *RCM* computes the hash value $M^*_{rc} = H(M_{rc}, Ts_{rc})$ and signs it using user's private key PrK_u, i.e., $S_{rc}\{M^*_{rc}\}$. *RCM* composes the package M_1 which has ID_u and ID_s in clear. At the resource server, *RAM* uses ID_u and ID_s to lookup for the corresponding user's certificate. *RAM* decrypts M_1 using its private key and verifies the digital signature using user public key PuK_u; it recomputes the message checksum and compares it with the one in M_1. Then, *RAM* validates ID_u and P_u using the information in the certificate.

- *User AT request* $[RAM \xrightarrow{M_2} ACM]$: if the user is authenticated, *RAM* forwards an encrypted and signed request package M_2 to the *ACM* server in order to issue the user's *authorization-token AT*. *RAM* forwards the *AT* request payload $M_{rs} = (ID_{rs}, ID_u, STZone, r, Ts_{rs})$, where Ts_{rs} is the payload's timestamp. *RAM* computes the hash value and signs it using its private key PrK_{rs}. Then, *RAM* encrypts the M_{rs} which is digitally signed $S_{rs}\{M^*_{rs}\}$ using *ACM* public key PuK_{as}. At the *Authorization Server*, *ACM* decrypts package M_2 and validates the digital signature using the public key of *RAM* PuK_{rs}. Nevertheless, if the user authentication fails, *RAM* sends access rejection response M_6 to the user without communicating with the *ACM* in order to reduce communication costs.
- *User AT response* $[ACM \xrightarrow{M_3} RAM]$: once message M_2 is authenticated, *ACM* issues the user's *AT* as requested in the M_3 package only if the user is authorized to activate the requested role. *ACM* maps the user ID_u and his/her current user $STZone$ with appropriate rights in the policy. Once the user's request is approved, *ACM* sends an encrypted and signed response acceptance package M_3 to *RAM*, which includes M_{as} payload as well as a digital signature $S_{as}\{M^*_{as}\}$ signed by ACM private key PrK_{as}. Payload $M_{as} = (ID_{as}, ID_{ut}, ID_u, AT, Ts_{as})$, has user's identifier ID_u, user's authorization-token AT, token identifier ID_{ut}, and timestamp Ts_{as}. However, if the requested role is not approved by the policy, then an access rejection response M_5 is sent to the *RAM* server.
- *Forwarding user AT* $[RAM \xrightarrow{M_4} RCM]$: after authenticating message M_3 from *ACM*, *RAM* stores a copy of the user's *authorization-token AT* in the *UATokens* list along with the token identifier ID_{ut}, user identifier ID_u, and device identifier ID_s. Subsequently, *RAM* forwards a signed and encrypted response package M_4 to *RCM*. The response package has payload $M_{rs} = (ID_{rs}, ID_u, ID_s, ID_{ut}, AT, Ts_{rs})$ as well as a digital signature $S_{rs}\{M^*_{rs}\}$ signed by the private key PrK_{rs}. Note that the user's *authorization-token* is bound to particular user ID_u and device ID_s. At user side, if M_4 package from *RAM* is authenticated, *RCM* stores the *authorization-token AT* in a secure element of the user's cell-phone. However, in case the access request is rejected by *ACM*, *RAM* directly forwards rejection response M_6 to *RCM*.

The basic resource access protocol thwarts a user from having conflicting roles or permissions. The SoD constraints are enforced by *ACM*. As shown in Algorithm 1, a user is not granted a new token for role r if it conflicts with at least one of the current user's active roles or if the current role has conflicting permissions.

5.3 Ongoing access protocol

The ongoing access protocol is responsible for revoking access once a user moves out of a valid *STZone*.

One choice was for the *STZone* information being periodically pushed to the *RAM* server, or the *RAM* server pulling the information from the users' mobile device. This incurs significant communication overhead, so we rejected this option. We adopted an alternative choice in which the user's device notifies the *RAM* server when the user fails to satisfy the *STZone* conditions specified in the *authorization-token*. In our solution, the *STZone Listener* component gets periodic updates by the *STZone Reader* component about the spatio-temporal coordinates of the user. Whenever the current user location or time does not satisfy the information in a user's *authorization-token*, the *STZone Listener* component revokes the user's *authorization-token* and requires *Request Builder* to send an access termination message to the *RAM* server. In order to implement the ongoing access protocol, two communications steps must be added to the protocol that we have described earlier. In Fig. 3, the communications messages M_7 and M_8 describe the additional exchanges needed to implement the ongoing access protocol.

- *Terminating user access* $[RCM \xrightarrow{M_7} RAM]$: *RCM* sends the termination access request M_7 to *RAM* at the time the current user *STZone* becomes invalid. It creates a termination message payload $M_{rc} = (ID_u, ID_s, ID_{ut}, P_u, Close, Ts_{rc})$ where the keyword "Close" indicates the termination of access. *RCM* concatenates M_{rc} with the user digital signature $S_{rc}\{M^*_{rc}\}$ and encrypts M_7 with *RAM* public key PuK_{rs}. The user's *authorization-token* is deleted at the client side in order to terminate the user access.
- *Revoking user privileges* $[RAM \xrightarrow{M_8} ACM]$: after authenticating the sender of M_7 package, *RAM* reads the keyword "Close" and then uses ID_u, ID_s, and ID_{ut} to lookup for the user's *authorization-token* in the $UATokens$ list and removes it. Therefore, if a user subsequently requests an access to a resource via deleted ID_{ut}, this request will be denied. *RAM* forwards an encrypted and signed termination request M_8 to *ACM* in order to update the current user's authorizations. The M_8 request has payload $M_{rs} = (ID_{rs}, ID_u, ID_{ut}, Close, Ts_{rs})$ as well as the digital signature $S_{rs}\{M^*_{rs}\}$ signed by *RAM* private key. At the authentication server, after authenticating the sender of M_8, the keyword "Close" indicates *ACM* to revoke the current user's active role and authorized permissions associated with user's *authorization-token* ID_{ut}.

5.4 Access suspension protocol

Since users are mobile, a user may momentarily depart the authorized location from which she is currently accessing some resources and come back after a short period of time. To maintain our design efficiency and ease of use, the user's privileges should not be permanently revoked, instead, these privileges must be frozen or suspended for the short period of time the user is off-site and reinstated after the user comes back on-site.

We achieve this design goal by modifying the communication steps M_7 of the ongoing resource access protocol in order to define the suspending resource access

protocol. In this protocol, *RCM* suspends user access and sends a resource access deferral package $M_7^{'}$ at the time a user unexpectedly leaves a current valid position. This message has format similar to M_7 except that the keyword "Close" is changed to "Freeze" to indicate that the user's privileges should be frozen for a certain time period.

A time window Tw is appended to $M_7^{'}$ package to determine the freezing time through which a user cannot exercise previously granted access rights. Once a user crosses back the boundary of the user's valid location and prior to the Tw expiration, she can automatically invoke earlier activated roles with no need to resend an activation request. At the resource server side, *RAM* reads the key word "Freeze" and instead of deleting user's *AT*, it prevents the user from exercising rights for the duration defined by the time Tw.

If *RAM* gets a resource access request from the user within the time window Tw, it gives the requested resource to the user and deletes Tw from the user's record. In case *RAM* does not hear anything from the user during the time period Tw, it forwards package M_8 to ACM in order to revoke the current user's active role and the authorized permissions combined with the user's authorization-token ID_{ut}.

6 Securing against some common attacks

Having defined our protocols, we must ensure that they are immune against some common attacks. Our attack analysis was inspired by some studies on threat models [14,36]. In the analysis of our system, we consider three types of attacks: the first one focuses on the software implementing the architecture components, the second one examines attacks on the messages exchanged between those components, and the third analyzes malicious entities pertaining to the protocol.

Attacks on software components

We assume that *RAM* and *ACM* components are implemented in computing machines equipped with appropriate software protection. For example, the servers must be protected by an appropriate firewall configuration and have malware protection. We also assume that the *RCM* component is installed in a phone's secure component, which can be accessed through proper access control. This is not unusual nowadays; Android platform provides a number of access control mechanisms, based on a Linux kernel, that protect access to shared data and functionality across multiple applications [4].

RAM and *ACM* may be targeted to denial-of-service attacks (DoS). Distribution and replication somewhat alleviate with this attack. We can counter these attacks somewhat by multicasting messages to multiple *RAM* and *ACM* servers. Thus, if one server is down, some other server can take over. However, with the distributed denial-of-service attack (DDoS attack), an attacker may succeed in compromising multiple servers. We assume the existence of sophisticated DDoS attack detection and mitigation techniques to counter such attacks.

Attacks on protocol messages

We are concerned about message confidentiality, message integrity, message authenticity, and message non-repudiation. Message confidentiality is provided by

encrypting each message with the public key of the recipient—this ensures that only the recipient can decrypt the message and read its contents. Message integrity is maintained by concatenating each message with the message digest and signing the message digest. Thus, even if an intruder modifies a message, he will not be able to change the digest. Consequently, any modification by an intruder can be detected. Each message is signed by the sender—this provides message authenticity.

Malicious entities

We are concerned about attackers impersonating as *RCM*, *RAM* and *ACM* servers. We are also concerned about the misuse caused by legitimate *RCMs* who may be over extending their privileges. We refer to these entities as *Malicious RCM*, *Malicious RAM*, and *Malicious ACM*.

Malicious RCM A *Malicious RCM* may want to perform various illegitimate activities in order to get unauthorized access to some resources. These are described below.

- *Eavesdropping*: An adversary may eavesdrop to get access to some sensitive information, such as, passwords and identifiers, in order to access some resource. All sensitive information is encrypted with the recipient's public key. Consequently, an adversary will not be able to eavesdrop on such information.
- *Modifications*: A modification attack occurs when an attacker intercepts and modifies the contents of communication messages in order to impersonate others to gain access. This attack is not applicable to our protocol because messages between *RCM* and *RAM* servers include a checksum that is digitally signed. Thus, any modification on messages can be easily detected.
- *Replay attacks*: In this attack, an eavesdropper intercepts authentication exchanges coming from a legitimate *RCM* and later replays that exchange in order to get access. Since the communication messages sent from *RCM* clients are encrypted and associated with unique timestamps, replay attacks will be detected. Moreover, the use of one-time passwords prevents an adversary from reusing passwords in the replayed messages.
- *Reflection attacks*: We assume that the various entities must first register themselves with a trusted *Certificate Authority*. The entities send messages to each other encrypted by the corresponding public keys. Moreover, messages contain nonce as well. Consequently, reflection attacks do not occur in our scheme.
- *Man-in-the-middle (MITM) attacks*: the use of public keys together with message specific timestamps and digital signatures prevents this from happening.
- *Illegitimate use*: A user cannot use the mobile device of another legitimate user to access resources. This is because the user needs to send his id, device id, and password to the *RAM* to request access. Clearly, an illegitimate user of the mobile device will not be able to provide them in order to be issued an *authorization-token*. Consequently, illegitimate users cannot make requests to get *authorization-tokens*.
- *Reusing authorization-tokens*: A user may want to access a resource using his previous *authorization-token*. However, when the *STZone* specified in the *authorization-token* expires, it gets deleted from the *RCM* and cannot be used.

- *Modifying authorization-tokens*: A user cannot modify the parameters in his *authorization-tokens* as they are stored in a secure tamper-proof component at the user's mobile device.
- *Separation of duty violations*: Separation of duty constraints prevents a user from activating two conflicting roles. Note that, *authorization token* is issued to the user only if he has no other conflicting roles activated by other *authorization tokens*. A user cannot obtain an *authorization-token* for a conflicting role from his friend either. This is because the authorization token is tied to a device as well as a user, so an *authorization-token* issued to his friend cannot be used on the user's device.

  In order to violate a separation of duty constraint, a user may try to request *authorization-tokens* for conflicting roles from two *RAM* servers. The *RAM* servers forward the request to the *ACM*s who are responsible for issuing the *authorization-tokens*. Since the *ACM*s are in a consistent state, *authorization-tokens* for conflicting roles will not be issued.

Impersonating RAM and ACM Since the entities are authenticated prior to the start of the protocol, an attacker cannot impersonate himself as a *RAM* or an *ACM* server. Moreover, all communications are encrypted with the public key of the recipient, so an attacker impersonating as *RAM* or *ACM* cannot decrypt those messages. Moreover, any message that is sent to the *RCM* must be digitally signed by the sender, so an attacker will not be able to send false messages to the *RCM*.

Our protocol also assumes that the sensitive data and algorithms are executed in a secure and tamper-proof component of the *RAM* or *ACM* servers. Thus, false generation and modification of *authorization-tokens* cannot happen in our protocol.

7 Formal analysis

In the previous section we made some informal arguments to demonstrate that our protocol is free from attacks. However, manual analysis is tedious and error-prone. Towards this end, we show how our protocol can be formally analyzed to give assurance that the attacks mentioned actually do not occur. We do not analyze all the attacks due to lack of space. We formalize and analyze only the man-in-the-middle attack.

Typically, the application is specified in UML [19] which is the de facto specification language used in the software industry. Recent works have advocated for using UML to specify access control policies [7,35] because it makes integration with the application and subsequent analysis easier. Unfortunately, UML does not have much tool support for checking behavior of systems. Consequently, security protocols cannot be verified directly using UML tools. Towards this end, we use Alloy [8] to check whether the resource access protocol described in the paper is vulnerable to the man-In-the-middle (MITM) attack. Alloy is a modeling language capable of expressing complex structural constraints and behavior. It has been successfully used to rigorously analyze security policies [1,3,50]. Researchers have also used Alloy for analyzing protocols and security properties [20,30,48]. Alloy is supported by an automated constraint solver called Alloy analyzer that searches instances of the model to check for satisfaction of system properties. The model is automatically translated into

a Boolean expression, which is analyzed by SAT solvers embedded within the Alloy analyzer. A user-specified scope on the model elements bounds the domain, making it possible to create finite Boolean formulas that can be evaluated by the SAT-solver. When a property does not hold, a counter example is produced that demonstrates how it has been violated.

In order to perform our analysis, the behavior specified in the UML design class model must be converted to an Alloy model that specifies behavioral traces. The UML-to-Alloy model transformation uses an intermediate model, namely, the snapshot transition model that provides a static description of behavior in terms of sequences of state transitions, where a transition represents an invocation of an operation described in the class diagram.

Yu et al. [51,52] describe how a software design class model with operation specifications can be transformed to a static model of behavior, called a snapshot transition model (STM). A snapshot represents a system object configuration at a particular time (i.e., a system state). A snapshot transition describes the behavior of an operation in terms of how system state changes after the invoked operation has completed its task. It consists of a before state, an after state, and the operation invocation that triggers the transition. An operation invocation is described by the operation name and the parameter values used in the invocation. Note that, protocol behavior can be represented as a sequence of state transitions, where each transition is triggered by an operation invocation. The snapshot transition model (STM) developed by Yu et al. is the intermediate model that we use in the UML-to-Alloy transformation. Once we have the translated Alloy specifications, it can be checked using the Alloy analyzer to check for attacks.

An Alloy model consists of *signature* declarations, *fields*, *facts* and *predicates*. Each signature consists of a set of *atoms* which are the basic entities in Alloy. Atoms are *indivisible* (they cannot be divided into smaller parts), *immutable* (their properties do not change) and *uninterpreted* (they do not have any inherent properties). Each field belongs to a signature and represents a relation between two or more signatures. A relation denotes a set of tuples of atoms. Facts are statements that define constraints on the elements of the model. Predicates are parameterized constraints that can be invoked from within facts or other predicates.

Figure 4 shows a fragment of an Alloy model for the resource access protocol shown in Fig. 3. The entire Alloy specification is given in Appendix A. Our communication protocol involves a set of types, namely, *ID*, *STZone*, *Role*, *Password*, *OnePassword*, *Timestamp*, *Module*, *Client*, *Server*, *Attacker*, *Key*, *PubKey*, *PriKey*, that are represented as signatures. Note that *Key* denotes an abstract type; we derive concrete types *PubKey* and *PriKey* from this basic type. Signature *Module* represents the participants and consists of an identifier *id* (instance of *ID*), public key *pubkey* (instance of *PubKey*), and private key *prikey* (instance of *PriKey*). *Client*, *Server*, and *Attacker* can be specialized from the base type *Module*. *Client*, derived from *Module*, has additional components, namely, role *r* (instance of *Role*), password *pwd* (instance of *Password*), one time password *onepwd* (instance of *OnePassword*), device identifier *ids* (instance of *ID*), and user's current spatio temporal zone *stzone* (instance of *STZone*). The type *Snapshot*, also represented as an Alloy signature, captures the state of the system at a given time and consists of *rcmclient* (instance of *Client*), *ramserver* (instance of *Server*), *acmserver* (instance of *Server*), and *attacker* (instance of *Attacker*).

```
abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig Server extends Module{}

sig Client extends Module{
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID,
stzone: one STZone
}

sig Attacker extends Client{}

sig Snapshot {
rcmclient: one Client,
ramserver: one Server,
acmserver: one Server,
attacker: one Attacker,
}
```

Fig. 4 Fragment of an Alloy model for the resource access protocol

(a)

```
pred M1[disj before, after: Snapshot,  senderPre,
senderPost: Client, receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey] {

senderPre = before.rcmclient
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.ramserver

// Frame condition
before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}
```

(b)

```
pred M4[disj before, after: Snapshot, senderPre,
senderPost: Module, receiverPre, receiverPost: Client,
idrsPre, idrsPost: ID, iduPre, iduPost: ID, idsPre,
idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre
receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.rcmclient

// Frame condition
before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}
```

Fig. 5 Alloy message predicates

(a)

```
pred ScenarioWithoutAttack{
let first = SnapshotSequence/first|
let second = SnapshotSequence/next[first] |
...
some rcmclient: Client| some ramserver: Module|
some disj idu, ids, idrs: ID| some r: Role|
...
M1[first, second, rcmclient, rcmclient, ramserver, ...] and
M4[second, third, ramserver, ramserver, rcmclient, ...]
}
```

(b)

```
pred ScenarioWithAttack{
let first = SnapshotSequence/first|
let second = SnapshotSequence/next[first]|
...
some rcmclient: Client| some ramserver: Module|
some attacker: Attacker| some disj idu, ids, idrs: ID|
...
Client2Attacker[first, second, rcmclient, rcmclient, attacker,...] and
Attacker2RAM[second, third, attacker, attacker, ramserver,...] and
RAM2Attacker[third, fourth, ramserver, ramserver, attacker,...] and
Attacker2Client[fourth, fifth, attacker, attacker, rcmclient, ...]
}
```

Fig. 6 Alloy simulation predicates

A communication protocol involves sending and receiving messages. We specify messages in our model using Alloy predicates. Figure 5 shows an example of an Alloy predicate that specifies the message $M1$ shown in the resource access protocol (see Fig. 3). In a communication protocol, we need to reason about the different states of the system resulting from sending and receiving messages. Recall that the state of a system is specified by the Alloy signature *Snapshot*. $M1$ predicate takes several parameters. The parameter *before* snapshot describes the system state before the message is sent and the parameter *after* snapshot describes a system state after the message has been sent. Parameter *senderPre* represents a message sender in the *before* snapshot while *senderPost* represents a message sender in the *after* snapshot. Similarly, *receiverPre*, *stzonePre*, *rPre*, *iduPre*, *idsPre*, and *pubkeyPre* represent states of objects accessed by the predicate in the *before* snapshot while *receiverPost*, *stzonePost*, *rPost*, *iduPost*, *idsPost*, and *pubkeyPost* represent states of objects in the *after* snapshot. The predicate specifies that the message sender must be a *RCM Client* and the message receiver must be a *RAM Server*. The sender's attributes must have the same value with the parameters of the predicate. For example, the value of the sender's current zone must be equal to the value of the $stzonePre$ parameter. The frame condition in the predicate specifies that the participants of the protocol must remain the same after the message has been sent.

Since it is assumed that *RAM* and *ACM* are trusted in the protocol, only the network communications between an *RCM Client* and a *RAM Server* are analyzed using the Alloy analyzer. In short, we analyze only messages $M1$ and $M4$ in Fig. 3. Figure 5 shows an Alloy predicate that specifies the $M4$ message from a *RAM Server* to a *RCM Client*.

Figure 6 shows an Alloy predicate that specifies a scenario without the MITM attack in which *RCM Client* sends $M1$ to *RAM Server* and then receives $M4$ from the *RAM Server*. The set of state transitions that occur due to these messages are captured through a sequence of snapshots, which we define as *SnapshotSequence*. Note that signature *SnapshotSequence* is predefined using an Alloy clause (i.e., "open util/ordering[Snapshot] as SnapshotSequence"). *util/ordering* is an Alloy utility function that takes as input a signature and declares a sequence of objects that are instances of the input signature. *first* is an Alloy function used to return the first element of a

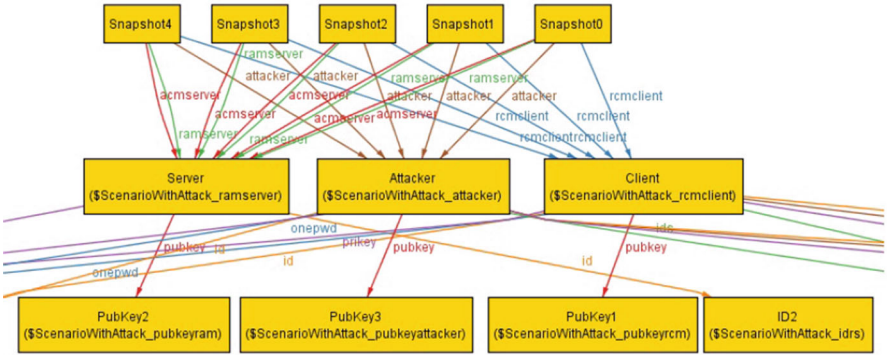

Fig. 7 A partial Alloy instance for the *ScenarioWithAttack* predicate shown in Fig. 6

sequence. *next* is an Alloy function that takes as input an element and returns the element next to the input element. Thus *SnapshotSequence/first* returns the first snapshot in the sequence. Thus *SnapshotSequence/next[first]* returns the second snapshot in the sequence. $first$ represents a system state before $M1$ has been sent, $second$ represents a system state after $M1$ has been sent and before $M4$ has been sent, and $third$ represents a system state after $M4$ has been sent. The Alloy analyzer uses the predicate in Fig. 6 to query whether there exists an instance that satisfies the predicate. For this scenario, the Analyzer did return an instance, indicating that the communication protocol between the *RCM Client* and the *RAM Server* was successfully simulated.

To verify whether the protocol is immune from an MITM attack, we introduce an attacker between a *RCM Client* and a *RAM Server*. The attacker makes independent connections with the *RCM Client* and the *RAM Server* respectively, and relays messages between them. In an MITM attack scenario, message $M1$ is replaced by two messages, $Client2Attacker$ and $Attacker2RAM$, where $Client2Attacker$ is a message from the client to the attacker and $Attacker2RAM$ is a message from the attacker to the server. Similarly, $M4$ is replaced by $RAM2Attacker$ and $Attacker2Client$, where $RAM2Attacker$ is a message from the server to the attacker and $Attacker2Client$ is a message from the attacker to the client. Figure 6 shows an Alloy predicate that simulates an MITM attack scenario. The Alloy analyzer uses the predicate in Fig. 6 to query whether there exists an instance that satisfies the predicate. For this MITM attack scenario, the Analyzer returned no instance, indicating that the MITM attack was not successfully simulated under the protocol described in the paper. Thus, the proposed protocol is immune from an MITM attack.

In order to test our protocol, we eliminated `senderPre.id = idrsPre` from $Attacker2Client$. The Alloy specification for this case is given in Appendix B. Without this statement, we can no longer verify the identity of the sender. The attacker can pose as the *RAM Server* causing the MITM attack to succeed. The Alloy analyzer in this case would return an instance, shown in Fig. 7, that demonstrates the MITM attack scenario. The Alloy instance consists of five snapshots, associated with the *ScenarioWithAttack* predicate. The *first* snapshot ($Snapshot0$) specifies an initial state before the $Client2Attacker$ message has been sent, the *second* ($Snapshot1$) repre-

sents the state after the $Client2Attacker$ message has been sent, the *third* snapshot ($Snapshot2$) specifies the state after the $Attacker2RAM$ message has been sent, the fourth snapshot ($Snapshot3$) represents a system state after the $RAM2Attacker$ message has been sent, and the fifth snapshot ($Snapshot4$) represents a system state after the $Attacker2Client$ message has been sent. The scenario provides an instance demonstrating MITM attack. Other attacks such as replay, reflection, etc, can also be analyzed using similar approaches.

8 Prototype implementation

We develop a proof-of-concept prototype for our architecture enforcing a spatio-temporal policy in mobile application. The prototype implements our design in a distributed system with multiple autonomous virtual servers. We implemented our clients and servers using Android.

We used Android because it is open source. Moreover, Google Android is built on the top of a Linux distribution that includes a Java virtual machine (JVM) designed to run on mobile devices [22]. Google Android platform has also many features for incorporating a GPS receiver package into mobile application code. Android has a powerful software development Kit (SDK) [21] that is easy-to-use in building mobile applications run on a variety of smartphones' technologies. The Android SDK provides a debugger, libraries, and a handset emulator that are necessary to write and test Android platform applications. The Android emulator allows us to simulate mobile applications before actual use. The development environment of our prototype is Eclipse IDE integrating the Android development tools (ADT) plug-in. For the first run of the Android application, we utilize the AVD manager in Eclipse to create a new Android device enhanced with a GPS receiver.

In order to implement our protocols, we used the source code from the *FlexiProvider* Toolkit [10] that has Java based cryptographic modules, including public key, digital signature, and *MD5* message digest. Prior to running the experiment, each entity should have its private key and the public keys of the entities that it will communicate with. We used the *KeyPairGenerator* class to produce a pair of public and private keys for each entity. The public keys are securely distributed using *AES* symmetric-key encryption. For the message authentication code, *RCM*, *RAM*, and *ACM* use the *MD5* that is supported in *FlexiProvider*.

In order to incorporate the location capabilities of Android into our prototype, we used the *LocationManager* package [21] to track the current user position and to capture the local time at which the location is retrieved. The *LocationManager.requestLocationUpdates* method revises the user's device's location periodically. A class implementing the *LocationListener* interface handles changes in the device location. The *LocationManager.getLastKnownLocation* method in the *LocationListener* implementer gets the last known location object which has the altitude, latitude, and longitude information.

The mobile application components are tested via the Android emulator and the test showed that our prototype works as expected. Our Android handset emulator in Fig. 8 displays the Android app component running in a user's mobile device. This

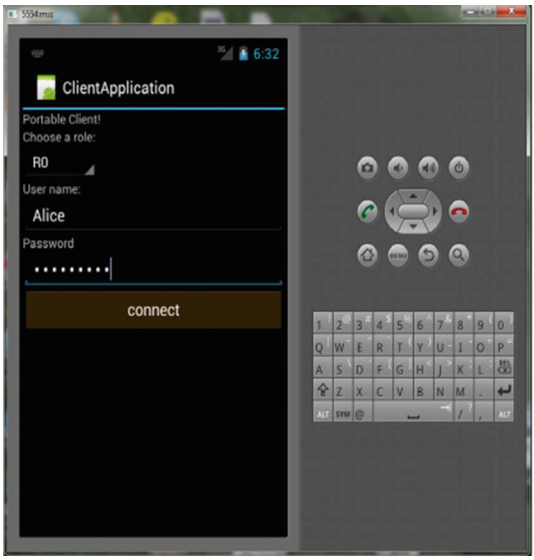

Fig. 8 Android handset emulator

handset emulator prompts a user to select his/her role and enters the user's identifier and password. Once the user enters these information and hits the connect button, the Android application software retrieves the last known user location and the local time. Then, it composes an access request package and sends it to one of the available *RAM Servers*.

In the experiment setup, only one machine is used to measure the back-end servers' overhead for processing access requests. The experiment is performed in one machine in order to eliminate the network delay. Furthermore, the time needed for the GPS receiver to get the location coordinates from the base station varies greatly and we wanted to eliminate this factor as well. The experiment is carried out in a machine with a Windows 7 platform running on Intel(R) Core(TM) 2 Duo CPU at 2.20 GH with 4.00 GB RAM.

The experiment evaluates the architecture performance on three virtual handset Android emulators, three *RAM* virtual servers, and three *ACM* virtual servers. These virtual machines communicate via traditional sockets. A virtual server running a local centralized MySQL database [39] is also instantiated on the same machine. The database server is accessible by the *RAM* and *ACM* servers in order to process users' requests. For each request, the handset emulator opens a new connection with one of the virtual *RAM* servers and closes the connection at the time it receives a response. The *RAM* server in turn opens a new connection with one of the *ACM* servers if the user login information is correct. The *RAM* server closes the connection when it gets a response from the endpoint *ACM* server.

To evaluate different spatio-temporal access scenarios, we have stored the logical locations and role names in two local files. Thus, for each request, the handset emulator randomly selects a location name and a role name from these files and sends them along with other information in the request package. This approach allows us to test

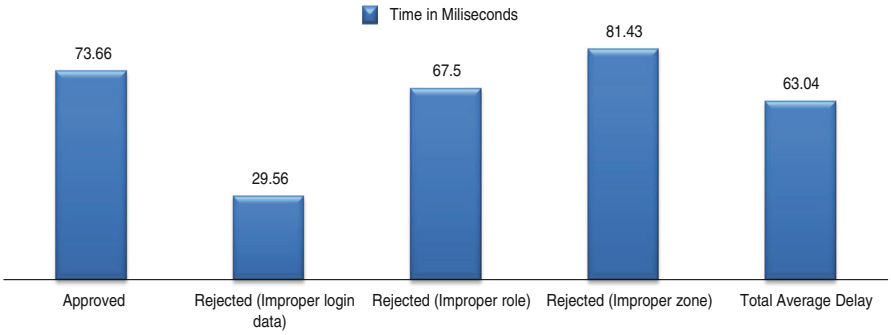

Fig. 9 Back-end average response delay plot

whether our application works as anticipated and ensures that correct policies have been specified. We measure the total time that elapses between issuing a new access request and getting the response. The response delay is evaluated using 150 requests sent simultaneously from three Android handset emulators. Each emulator sends 50 requests. The responses vary based on the information in the request packages. For example, a request is approved if and only if the login information is correct, the requested role can be authorized, and the current user's zone is acceptable. Otherwise, the request is rejected.

The results in Fig. 9 show the average response delay for each response type as well as the total average delay in milliseconds. The overall delay yielded by the basic resource access protocol is 63.04 ms. Consequently, implementing our architecture for enforcing spatio-temporal policies is indeed viable. The rejected requests due to invalid login information yields 29.56 ms, which is the smallest response delay because *RAM* servers send these responses immediately without consulting the *ACM* servers.

9 Conclusion and future work

In this work we proposed an enforcement mechanism of a spatio-temporal RBAC in mobile applications. We proposed an architecture for a mobile system enforcing our spatio-temporal model. We developed a number of protocols that consider spatio-temporal information for initiating and maintaining access under different circumstances. We also demonstrated how our protocols can be analyzed using Alloy to provide assurance that they are indeed free from attacks. We also implemented our protocol and gave some results indicating the performance penalty of our approach.

A lot of work remains to be done. Before such a technology can be deployed, we need to do experiments to measure the impact of GPS devices on battery consumption. We also need to accurately measure the delays in the various types of networks. Our promising direction of a future work is to extend our spatio-temporal access control model for workflows which consists of a set of tasks that are coordinated by control-flow, data-flow and temporal dependencies. It would be interesting to see how these various dependencies interact with the spatio-temporal constraints of the workflow.

Our future work also includes deploying this model for a real-world healthcare dengue decision support system (DDSS). We also plan to provide a more flexible spatio-temporal access control that is able to make authorization decisions in the presence of uncertainty, which is possible if the user's location cannot be accurately determined. Such a scenario can occur when a user is on the move and his trajectory is used to make access control decisions.

Appendix A

Alloy specification of the access control protocol

```
module SecurityProtocol

open util/ordering[Snapshot] as SnapshotSequence

abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Snapshot {

rcmclient: one User,
ramserver: one Module,
acmserver: one Module,

attacker: one Module,
}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig User extends Module{
stzone: one STZone,
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID
}

sig Attacker extends User{}

pred M1[disj before, after: Snapshot, senderPre, senderPost: User,
receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
```

```
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.rcmclient
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Client2Attacker[disj before, after: Snapshot,
senderPre, senderPost: User, receiverPre, receiverPost: Attacker,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.rcmclient
receiverPre = before.attacker

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2RAM[disj before, after: Snapshot,
senderPre, senderPost: Attacker, receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
```

```
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred M4[disj before, after: Snapshot, senderPre, senderPost: Module,
receiverPre, receiverPost: User,
idrsPre, idrsPost: ID, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}


pred RAM2Attacker[disj before, after: Snapshot, senderPre,
senderPost: Module, receiverPre, receiverPost: Attacker,
idrsPre, idrsPost: ID, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.attacker

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
```

```
before.attacker = after.attacker
}

pred Attacker2Client[disj before, after: Snapshot,
senderPre, senderPost: User, receiverPre, receiverPost: Attacker,
idrsPre, idrsPost: ID, iduPre, iduPost: ID,

idsPre, idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.rcmclient

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre
receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Scenario1{
let first = SnapshotSequence/first|let second = SnapshotSequence/next[first] |
let third = SnapshotSequence/next[second]| some rcmclient: User|
some ramserver: Module| some disj idu, ids, idrs: ID|
some r: Role| some stzone: STZone| some disj pubkeyrcm, pubkeyram: PubKey|
M1[first, second, rcmclient, rcmclient, ramserver,
ramserver, stzone, stzone, r, r, idu, idu, ids, ids, pubkeyram, pubkeyram] and
M4[second, third, ramserver, ramserver, rcmclient,
rcmclient, idrs, idrs, idu, idu, ids, ids, pubkeyrcm, pubkeyrcm]
}

run Scenario1 for 3

pred Scenario2{
let first = SnapshotSequence/first|let second = SnapshotSequence/next[first]|
let third = SnapshotSequence/next[second]|
let fourth = SnapshotSequence/next[third]|
let fifth = SnapshotSequence/next[fourth]| some rcmclient: User|
some ramserver: Module| some attacker: Attacker|
some disj idu, ids, idrs: ID| some r: Role| some stzone: STZone|
some disj pubkeyrcm, pubkeyram, pubkeyattacker: PubKey|
Client2Attacker[first, second, rcmclient, rcmclient,
attacker, attacker, stzone, stzone, r, r, idu, idu, ids, ids,
pubkeyattacker, pubkeyattacker] and
Attacker2RAM[second, third, attacker, attacker,
ramserver, ramserver, stzone, stzone, r, r, idu, idu, ids, ids,
pubkeyram, pubkeyram] and
RAM2Attacker[third, fourth, ramserver, ramserver,
attacker, attacker, idrs, idrs, idu, idu, ids, ids, pubkeyattacker,
  pubkeyattacker] and
Attacker2Client[fourth, fifth, attacker, attacker,
rcmclient, rcmclient, idrs, idrs, idu, idu, ids, ids, pubkeyrcm, pubkeyrcm]
```

```
}

run Scenario2 for 5
run Attacker2Client
```

Appendix B

Alloy specification of successful MITM attack

```
module SecurityProtocol

open util/ordering[Snapshot] as SnapshotSequence

abstract sig Key{}
sig PubKey, PriKey extends Key{}

sig ID{}
sig STZone{}
sig Role{}
sig Password{}
sig OnePassword{}
sig Timestamp{}

sig Module{
id : one ID,
pubkey: one PubKey,
prikey: one PriKey,
}

sig Server extends Module{}

sig Client extends Module{
r: one Role,
pwd: one Password,
onepwd: one OnePassword,
ids: one ID,
stzone: one STZone
}

sig Attacker extends Client{}

sig Snapshot {
rcmclient: one Client,
ramserver: one Server,
acmserver: one Server,
attacker: one Attacker,
}

pred Client2Attacker[disj before, after: Snapshot,
senderPre, senderPost: Client, receiverPre, receiverPost: Attacker,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.rcmclient
```

```
receiverPre = before.attacker

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.rcmclient
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2RAM[disj before, after: Snapshot,
senderPre, senderPost: Attacker, receiverPre, receiverPost: Module,
stzonePre, stzonePost: STZone, rPre, rPost: Role,
iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.ramserver

senderPre.stzone = stzonePre
senderPre.r = rPre
senderPre.id = iduPre
senderPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.ramserver

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred RAM2Attacker[disj before, after: Snapshot,
senderPre, senderPost: Module, receiverPre, receiverPost: Attacker,
idrsPre, idrsPost: ID, iduPre, iduPost: ID, idsPre, idsPost: ID,
pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.ramserver
receiverPre = before.attacker

senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.ramserver
```

```
receiverPost = after.attacker

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}

pred Attacker2Client[disj before, after: Snapshot,
senderPre, senderPost: Client, receiverPre, receiverPost: Attacker,
idrsPre, idrsPost: ID, iduPre, iduPost: ID,
idsPre, idsPost: ID, pubkeyPre, pubkeyPost: PubKey]{

senderPre = before.attacker
receiverPre = before.rcmclient

// This is a key to fix the MITM attack because the
// client expects a message containing the server's identity, while
// senderPre.id = idrsPre in the predicate ensures that
// the client will receive an identity from the sender, which is
// the attacker in the MITM attack scenario.
// senderPre.id = idrsPre

receiverPre.id = iduPre
receiverPre.ids = idsPre

receiverPre.pubkey = pubkeyPre

senderPost = after.attacker
receiverPost = after.rcmclient

before.rcmclient = after.rcmclient
before.ramserver = after.ramserver
before.acmserver = after.acmserver
before.attacker = after.attacker
}


pred ScenarioWithAttack{
let first = SnapshotSequence/first|let second = SnapshotSequence/next[first]|
let third = SnapshotSequence/next[second]|let fourth =
SnapshotSequence/next[third]|
let fifth = SnapshotSequence/next[fourth]| some rcmclient: Client|
some ramserver: Module| some attacker: Attacker|
some disj idu, ids, idrs: ID| some r: Role| some stzone: STZone|
some disj pubkeyrcm, pubkeyram, pubkeyattacker: PubKey|
Client2Attacker[first, second, rcmclient, rcmclient,
attacker, attacker, stzone, stzone, r, r, idu, idu, ids,
ids, pubkeyattacker, pubkeyattacker] and
Attacker2RAM[second, third, attacker, attacker, ramserver,
ramserver, stzone, stzone, r, r, idu, idu, ids, ids, pubkeyram, pubkeyram] and
RAM2Attacker[third, fourth, ramserver, ramserver,
attacker, attacker, idrs, idrs, idu, idu, ids, ids,
pubkeyattacker, pubkeyattacker] and
Attacker2Client[fourth, fifth, attacker, attacker,
rcmclient, rcmclient, idrs, idrs, idu, idu, ids, ids, pubkeyrcm, pubkeyrcm]
}

run ScenarioWithAttack for 5
```

Appendix C

Spatio-temporal access control model specification

We present our complete model and formalize its specification using the unified modeling language (UML) and the object constraint language (OCL).

Effect of spatio-temporal constraints on RBAC entities

The RBAC entities: users, roles, permissions, and objects are associated with spatio-temporal zones.

Users We assume that each valid user, interested in doing some location-sensitive operations, carries a locating device that is able to track his location. The location of a user changes with time. The spatio-temporal zone associated with a user gives the user's current location and time.

Note that, time and location can have different levels of granularity. For example, the current time can be expressed as 12:00:05 or 12:00 pm. Similarly, a user's current location can be Fort Collins or it can be Colorado. The user's current location and time information will be used for making access decisions. Consequently, we require the minimal temporal and location be used to express the spatio-temporal zone associated with a user. We define the function $currentzone$ that returns the minimal spatio-temporal zone associated with a user. This function is formally defined as follows:

– $currentzone : Users \rightarrow STZones$

Objects Objects may also be mobile like the user. Here again, we have locating devices that track the location of an object. Moreover, an object may not be accessible everywhere and anytime. For example, tellers can only review customer information at a teller office during working hours. The *ozones* function returns the spatio-temporal zones that determine where and when every object is available.

– $ozones : Objects \rightarrow 2^{STZones}$

Roles Role can be assigned or activated only in specific locations and time. The role of on-campus student can only be assigned/activated inside the campus during the semester. The spatio-temporal zone associated with a role gives the location and time from which roles can be assigned or activated. The $rzones$ function gives the set of spatio-temporal zones associated with a given role.

– $rzones : Roles \rightarrow 2^{STZones}$

Permissions Permissions are also associated with a spatio-temporal zone that indicate where and when a permission can be invoked. For example, a permission to perform a backup of servers can be executed only from the department after 10 pm on Friday nights. The function $pzones$ gives the zones in which a specific permission can be accessed.

– $pzones : Permissions \rightarrow 2^{STZones}$

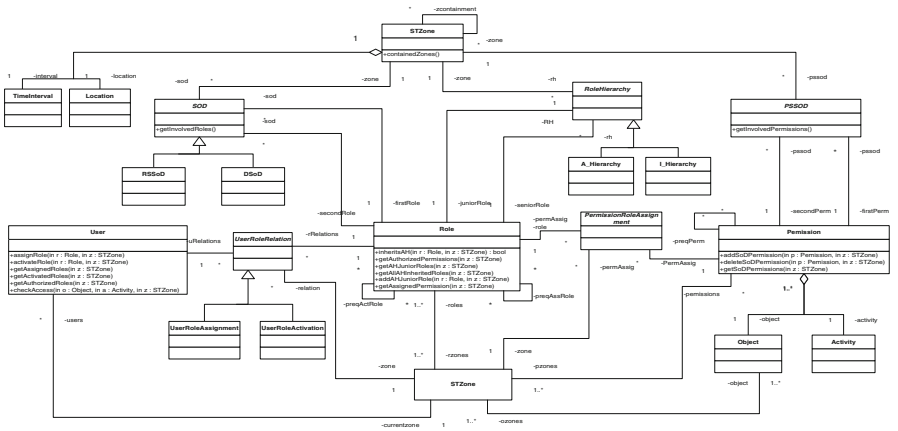

Fig. 10 UML class model for STRBAC

Figure 10 shows the class diagram of GSTRBAC. Therefore, a security policy of a mobile application can be specified as one possible instance of this GSTRBAC class diagram. The GSTRBAC entities: *User*, *Role*, *Permission*, *Object*, *Activity*, and *STZone*, are represented by classes. *Permission* is represented in the GSTRBAC class diagram as an aggregation of the classes *Object* and *Activity*. The *STZone* class aggregates the location and time subclasses. In *STZone* class, *zcontainment* is a reflexive association specifying that a zone can contain other zones. Different relationships between entities including *UserRoleAssignment*, *UserRoleActivation*, *PermissionRoleAssignment*, *RoleHierarchy*, and *SoD* are modeled using association classes which are transformed to normal classes following the modeling guidelines in [19,28]. These association classes have binary relationships with *STZone* class to enforce the spatio-temporal constraints.

Effect of spatio-temporal constraints on RBAC operations

User-role assignment A user-role assignment is location and time dependent. That is, a user can be assigned to a role provided the user is in specific locations. For example, a person can be assigned the on-campus student role only when he is in the campus during the semester. This requirement is expressed using the zone concept:

– $UserRoleAssignment \subseteq users \times roles \times STZones$

This relationship is depicted in the GSTRBAC class diagram as association class *UserRoleAssignment*. The OCL operation *assignRole* assigns role r to user u in zone z if z is in the set of *rzones*, user u is present in zone z, and role r is not already assigned to user u in zone z. For the lack of space, we omit the descriptions of the OCL queries used in the *assignRole* operation.

```
context User::assignRole(r: Role, z:STZone):
UserRoleAssignment
pre: r.rzones->includes(z)
```

```
pre: z.containedZones()->includes(self.
currentzone)
pre: self.getAssignedRoles(z)->excludes(r)
post: self.getAssignedRoles(z)->includes(r)
```

User-role activation A user can activate a role if the role can be activated on the specific zone and it is already assigned to that user. For example, the role of doctor trainee can only be activated in a hospital during the training period. We define the *UserRoleActivation* relation to determine the current active roles based on zones:

– $UserRoleActivation \subseteq users \times roles \times STZones$

In GSTRBAC class diagram, *UserRoleActivation* class is specified in a manner similar to *UserRoleAssignment*. The only difference is that the *activateRole* operation ensures that a user is already assigned to a role before the role is being activated.

Check access This operation checks whether a user is authorized to perform some operation on an object during a certain time and from a certain location. A user is allowed to fire a missile if he is assigned the role of top secret commander and he is in the controller room of the missile during a severe crisis period. Thus, a user can access an object in a certain zone if that user has activated a role which has an appropriate permission for that object in that zone.

```
context User::checkAccess(o:Object,a:
Activity,z:STZone):Boolean
post: result = getActivatedRoles(z)->
collect( r | r.getAuthorizedPermissions(z))
->asSet()->
exists( p | p.object=o and p.activity=a and
 o.ozones->includes(z))
```

Permission-role assignment Permissions can only be assigned to a role during specific time and locations. For example, the permission of opening a cashier drawer in a store should be only assigned to a salesman role during the day time. The assignment of permissions to roles is specified based on zones.

– $PermissionRoleAssignment \subseteq permissions \times roles \times STZones$

The following OCL operation assigns permission p to role r in zone z if z is in the set of $pzones$ and $rzones$.

```
context Role::assignPermission(p:Permission,
z:STZone): PermissionAssignment
pre: p.pzones->includes(z) and self.rzones->
includes(z)
pre: self.getAssignedPermissions(z)->excludes(p)
post: self.getAssignedPermissions(z)->includes(p)
```

Spatio-temporal role hierarchy

The permission-inheritance hierarchy (*I-Hierarchy*) and the role-activation hierarchy (*A-Hierarchy*) are two variations of role hierarchy (*RoleHierarchy*) in RBAC [41,27].

In our model, a senior role could have a subset of junior roles in a particular zone. The spatio-temporal role hierarchies are formally defined as follows:

- *RoleHierarchy* $\subseteq$ *Roles* $\times$ *Roles* $\times$ *STZones*
- *I-Hierarchy* $\subseteq$ *RoleHierarchy*, *A-Hierarchy* $\subseteq$ *RoleHierarchy*, and *I-Hierarchy* $\cap$ *A-Hierarchy* = ϕ

The subtypes of *RoleHierarchy* are represented in the GSTRBAC class diagram by the subclasses *I-Hierarchy* and *A-Hierarchy*, which are connected to *STZone* class to restrict the roles associations.

Permission-inheritance hierarchy In permission-inheritance hierarchy, a senior role r can only inherit junior role r' permissions in zone z if both roles are available in zone z. A project manager inherits the permissions of a developer when he is at the customer site giving a demo. The following OCL expression specifies the spatio-temporal constraint on *I-Hierarchy* for adding new junior role.

```
context Role::addIHJuniorRole(r:Role,z:STZone):
I_Hierarchy
pre: self.rzones->includes(z) and r.rzones->
includes(z)
pre: self.getIHJuniorRoles(z)->excludes(r)
post: self.getIHJuniorRoles(z)->includes(r)
```

The delete operation of a junior role in *I-Hierarchy* can be defined in the similar manner. The *I-Hierarchy* relationship is acyclic as shown by the following OCL constraint.

```
context r1,r2: Role
inv IHierarchy_Cycle_Constraint: not
STZone.allInstances-> exists(z|r1.inheritsIH(r2,z)
and r2.inheritsIH(r1,z)and r1<>r2)
```

The boolean operation *inheritsIH(r,z)* returns true if role r is directly or indirectly a junior role of the context role in particular zone, otherwise it returns false.

```
inheritsIH(r:Role,z:STZone): Boolean =
if (self.getIHJuniorRoles(z)->includes(r))
then true
else self.getIHJuniorRoles(z)->
exists(j | j.inheritsIH(r,z)) endif
```

We define the OCL query operation *getAuthorizedPermissions(z)* to get the authorized permissions for a given role at zone z through direct assignment or indirect *I-Hierarchy*.

```
context Role::getAuthorizedPermissions(z:STZone):
Set(Permission)
Post: result= self.getAssignedPermissions(z)->
union(self.getAllIHInheritedRoles(z)->collect(r |
r.getAssignedPermissions(z)))->asSet()
```

Role-activation hierarchy Restricted spatio-temporal *A-Hierarchy* allows members of senior roles to activate junior roles in predefined spatio-temporal zones. For example, a department chair can activate a staff role during the semester inside the department building. The OCL operations of adding and deleting junior roles to the *A-Hierarchy* are defined in similar manner to *I-Hierarchy*. Further, the acyclic constraints on *A-Hierarchy* is enforced in the same way of the *I-Hierarchy*.

The only differences are that, the OCL query operation *getAHJuniorRoles(z)* returns all the junior role in *A-Hierarchy* of the context role in particular zone. Moreover, the OCL query operation *getAuthorizedRoles(z)* gives the authorized activation roles for the context user that are either explicitly assigned or implicitly obtained through *A-Hierarchy* in certain zones.

```
context User:: getAuthorizedRoles(z:STZone):
Set(Role)
post: result= self.getAssignedRoles(z)->
union(self.getAssignedRoles(z)->collect(r|
r.getAllAHInheritedRoles(z))->
asSet())
```

Spatio-temporal separation of duty

The static SoD (SSoD) and dynamic SoD (DSoD) are two special classes of the SoD constraints in RBAC [17]. Further, the role SSoD (RSSoD) constraints are defined on roles assignment, while the permission SSoD (PSSoD) constraints are defined on permissions assignments.

In our model, the conflicting roles and permissions in SoD are defined over some zones. The spatio-temporal RSSoD, PSSoD, and DSoD relations are formally defined as follows:

- $RSSoD \subseteq Roles \times Roles \times STZones$
- $DSoD \subseteq Roles \times Roles \times STZones$, and $RSSoD \cap DSoD = \phi$
- $PSSoD \subseteq permissions \times permissions \times STZones$

The static and dynamic SoD relations are represented in the GSTRBAC class diagram using the associations classes RSSoD, PSSoD, and DSoD, which connect the conflicting entities with certain zones.

Role SSoD The same individual should not be assigned to specific roles in specific location for some duration. For example, the same user should not be assigned to billing clerk and account receivable clerk roles in the same time at specific trade corporation. The following OCL invariant forbids the assignment of conflicting roles in a particular zone.

```
context User
inv RSSOD_Constaint: STZone.allInstances->forAll( z |
not self.getAssignedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2)))
```

However, the above constraint might be violated through role hierarchy relation. For example, a billing supervisor role might be a senior role of the two conflicting roles billing clerk and account clerk at the same time and in the same accounting department. The following OCL constraint prevents such situation.

```
context User
inv RSSOD_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedRoles(z)->
exists(r1,r2 | r1.getSSoDRoles(z)->includes(r2)))
```

Permissions SSoD PSSoD prevents the assignment of conflicting permissions to a role. For example, a loan officer is not permissible to issue loan request and approve it in the bank building during the day-time. The following OCL invariant expresses the PSSoD requirement in our model.

```
context Role
inv PSSOD_Constaint: STZone.allInstances->
forAll( z | not self.getAssignedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2)))
```

However, this constraint might be violated through *I-Hierarchy* in which a senior role inherits some junior roles that have been mutually assigned conflicting permissions. The following OCL invariant prevents the violation of PSSoD via *I-Hierarchy*.

```
context Role
inv PSSOD_RH_Constraint: STZone.allInstances->
forAll( z | not self.getAuthorizedPermissions(z)->
exists(p1,p2 | p1.getPSSoDPermissions(z)->
includes(p2)))
```

DSoD Two conflicting activation roles cannot be activated in some spatio-temporal zones by the same user. For example, the simultaneous activation of cashier and cashier supervisor is forbidden during the working hours in the same store to deter such user from committing a fraud. The DSoD constraints are expressed in OCL invariants in a similar manner to the RSSoD constraints. The only difference is that the OCL invariants prevent the activations of conflicting roles that are connected by DSoD in some zones through either the explicit role assignment or the implicit *A-Hierarchy*.

Spatio-temporal prerequisite constraints

In RBAC, the prerequisite constraints obligates that some actions to be taken prior to performing an operation [16].

Prerequisite constraints on user-role assignment The prerequisite constraint on roles assignments imposes that a user must be assigned to some less critical roles in a given spatio-temporal zone before being assigned more critical roles in specific zones. For example, the role of emergency-nurse can be assigned to John in the urgent care

unit from 12:00 to 5:00 am if he is assigned the role of nurse-on-night-duty at the hospital during those hours. The following OCL invariant expresses the prerequisite constraints on user-role assignment. The query operation *getPreqAssRoles()* returns all the assignment prerequisite roles needed for assigning a certain role.

```
context User
inv Prerequiste_URAssign: STZone.allInstances->
forAll(z | Role.allInstances->
forAll(r1 | (self.getAssignedRoles(z)->
includes(r1)) implies (self.getAssignedRoles(z)->
includesAll(r1.getPreqAssRoles())))))
```

Prerequisite constraints on permission-role assignment The prerequisite constraints on permissions assignments indicates that a role can be assigned a permission in a specific zone if some prerequisite permissions are already assigned to that role in the same zone. For example, a bank teller must have the permission of reading an account during working hours before he can be given the permission to update that account. The prerequisite constraint on permission-role assignment can be specified using OCL expression as follows.

```
context Role
inv Prerequist_PRAssign: STZone.allInstances->
forAll(z | Permission.allInstances->
forAll(p1 | (self.getAssignedPermissions(z)->
includes(p1)) implies
(self.getAssignedPermissions(z) ->
includesAll(p1.getPrerequisitePermissions())))))
```

Prerequisite user-role activation A role can be activated if some prerequisite roles are already activated in specific zones. For example, in a university the teaching assistant role can be activated during a semester in a department if the student role can be activated during the same time. This requirement is specified in our model in the same way of the prerequisite user-role assignment constraint except that the OCL query *getPreqAssRoles()* is substituted with *getPreqActRole()*. The query operation *getPreqActRole()* returns all activation prerequisite roles needed to activate a role.

References

1. Schaad A, Moffett J (2002) A lightweight approach to specification and analysis of role-based access control extensions. In: Proceedings of the symposium on access control models and technologies (SACMAT), pp 13–22
2. Anne A (2004) XACML profile for role-based access control (RBAC). OASIS Access Control TC Comm Draft 1:13
3. Samuel A, Ghafoor A, Bertino E (2007) A framework for specification and verification of generalized spatio-temporal role based access control model. Technical report CERIAS TR 2007–08, Purdue University, West Lafayette
4. Chaudhuri A (2009) Language-based security on Android. In: Proceedings of the ACM workshop on programming languages and analysis for security (PLAS), pp 1–7

5. Shafiq B, Masood A, Joshi J, Ghafoor A (2005) A role-based access control policy verification framework for real-time systems. In: Proceedings of the workshop on object-oriented real-time dependable systems (WORDS), pp 13–20
6. Bose B, Sane S (2010) DTCOT: distributed timeout based transaction commit protocol for mobile database systems. In: Proceedings of the international conference and workshop on emerging trends in technology (ICWET), Mumbai, India, pp 518–523
7. Kim D-K, Ray I, France RB, Li N (2004) Modeling role-based access control using parameterized UML models. In: Proceedings of the 7th international conference FASE'2004, pp 180–193
8. Daniel J (2002) Alloy: a lightweight object modelling notation. ACM Trans Softw Eng Methodol 11(2):256–290
9. Daniel M, Gerald P, Richard M (1980) A locking protocol for resource coordination in distributed databases. ACM Trans Database Syst 5(2):103–138
10. Technische Universität Darmstadt. FlexiProvider. http://www.flexiprovider.de/overview.html/. Accessed on 30 Nov 2012
11. Bertino E, Catania B, Damiani ML, Perlasca P (2005) GEO-RBAC: a spatially aware RBAC. In: Proceedings of the ACM symposium on access control models and technologies (SACMAT), pp 29–37
12. Bertino E, Piero B, Elena F (2001) TRBAC: a temporal role-based access control model. ACM Trans Inf Syst Secur 4(3):191–233
13. Sposaro F, Tyson G (2009) iFall: an Android application for fall monitoring and response. In: Proceedings of the annual international conference of the IEEE at Engineering in Medicine and Biology Society (EMBC), 3–6 Sept 2009, pp 6119–6122
14. Frank S, Window S (2004) Threat modeling (Microsoft professional). Microsoft Press, Redmond (ISBN: 0735619913)
15. Hansen F, Oleshchuk V (2003) SRBAC: a spatial role-based access control model for mobile systems. In: Proceedings of the 8th Nordic workshop secure IT systems (NORDSEC), pp 129–141
16. Ahn G, Shin M (2001) Role-based authorization constraints specification using object constraint language. In: Proceedings of the IEEE international workshops on enabling technologies: infrastructure for collaborative enterprises (WETICE), pp 157–162
17. Gail-Joon A, Ravi S (2000) Role-based authorization constraints specification. ACM Trans Inf Syst Secur 3(4):207–226
18. US Government (2012) Global positioning system. http://www.gps.gov/. Accessed on 30 Nov 2012
19. Booch G, James R, Ivar J (2005) The unified modeling language user guide, 2nd edn. Addison-Wesley Professional, Boston
20. Grisham P, Chen C, Khurshid S, Perry D (2006) Design and validation of a security model with the Alloy analyzer. In: Proceedings of the workshop at ACM SIGSOFT first Alloy, 6th Nov 2006, Portland, OR, USA
21. Google Inc. (2012) Android SDK. http://developer.android.com/sdk/index.html. Accessed on 30 Nov 2012
22. Google Inc. (2012) The Android mobile (OS). http://www.android.com/. Accessed on 30 Nov 2012
23. Ray I, Kumar M, Yu L (2006) LRBAC: a location-aware role-based access control model. In: Proceedins of the 2nd international conference on information systems security (ICISS 2006), 17–21 Dec 2006, Indian Statistical institute, Kolkata, India, pp 147–161
24. Ray I, Toahchoodee M (2007) A spatio-temporal role-based access control model. In: Proceedings of the DBSec, pp 211–226
25. Jaehong P, Ravi S (2004) The UCON_{ABC} usage control model. ACM Trans Inf Syst Secur 7(1):128–174
26. James J, Elisa B, Usman L, Arif G (2005) A generalized temporal role-based access control model. IEEE Trans Knowl Data Eng 17(1):4–23
27. James J, Elisa B, Usman L, Arif G (2005) A generalized temporal role-based access control model. IEEE Trans Knowl Data Eng 17(1):4–23
28. Larman C (2004) Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3rd edn. Prentice Hall, Englewood Cliffs

29. Chen L, Crampton J (2008) On spatio-temporal constraints and inheritance in role-based access control. In: Proceedings of the ACM symposium on information, computer and communications security (ASIACCS), Mar 2008, pp 205–216
30. Lin A, Bond M, Clulow J (2007) Modeling partial attacks with Alloy. In: Proceedings of the workshop on security protocols, pp 20–33
31. Lockhart H, Parducci B, Levinson R (2012) OASIS eXtensible access control markup language (XACML) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml/. Accessed on 30 Nov 2012
32. Tamer Özsu M, Valduriez P (1999) Principles of distributed database systems, 2nd edn. Prentice-Hall, Englewood cliffs (ISBN-10: 1441988335)
33. Toahchoodee M, Ray I (2011) On the formalization and analysis of a spatio-temporal role-based access control model. J Comput Secur 19(3):399–452
34. Toahchoodee M, Ray I, Anastasakis K, Georg G, Bordbar B (2009) Ensuring spatio-temporal access control for real-world applications. In: Proceedings of the 13th ACM symposium on access control models and technologies (SACMAT), Estes Park, CO, USA, 11–13 June 2008 pp 13–22
35. Manuel K, Francesco P-P (2006) UML specification of access control policies and their formal verification. Softw Syst Modell 5(4):429–447
36. Michael H, David L (2002) Writing secure code, 2nd edn. Microsoft Press, Redmond (ISBN: 0735617228)
37. Kirkpatrick M, Bertino E (2010) Enforcing spatial constraints for mobile RBAC systems. In: Proceedings of the 15th ACM symposium on access control models and technologies (SACMAT), Pittsburgh, pp 99–108
38. Xu M, Wijesekera D (2009) A role-based XACML administration and delegation profile and its enforcement architecture. In: Proceedings of the 6th ACM workshop on secure web services (SWS), 13 Nov 2009, Chicago, IL, USA, pp 53–60
39. MySQL (2012) The world's most popular open source database. http://www.mysql.com/. Accessed on 30 Nov 2012
40. Abdunabi R, Al-Lail M, Ray I, Robert B (2013) Specification, validation, and enforcement of a generalized spatio-temporal role-based access control model. IEEE Syst J (to be appear)
41. Ravi S, Edward C, Hal F, Charles Y (1996) Role-based access control models. IEEE Comput 29(2):38–47
42. Ravi S, Kumar R, Xinwen Z (2006) Secure information sharing enabled by trusted computing and PEI models. In: Proceedings of the ACM symposium on information, computer and communications security (ASIACCS'06), 21–24 Mar 2006, Taipei, Taiwan
43. Mondal S, Sural S (2008) Security analysis of temporal-RBAC using timed automata. In: Proceedings of the 4th international symposium on information assurance and security (IAS), 8–10 Sept 2008, pp 37–40
44. Ravi S (1995) Rationale for the RBAC96 family of access control models. In: Proceedings of the 1st ACM workshop on role-based access control
45. Subhendu A, Samrat M, Shamik S, Arun M (2009) Role based access control with spatiotemporal context for mobile applications. Trans Comput Sci 4:177–199
46. Subhendu A, Shamik S, Arun M (2007) STARBAC: spatio temporal role based access control. In: Proceedings of the OTM, pp 1567–1582
47. Syed A, Mohammad I (2011) Location-based services handbook: applications, technologies, and security. CRC Press, Boca Raton (ISBN: 1420071963)
48. Taghdiri M, Jackson D (2003) A lightweight formal analysis of a multicast key management scheme. In: Proceedings of the FORTE, pp 240–256
49. Arensman W, Whipple J, Boler M (2009) A public safety application of GPS-enabled smartphones and the Android operating system. In: Proceedings of the systems, man and cybernetics (SMC), pp 2059–2061
50. Sun W, France R, Ray I (2011) Rigorous analysis of UML access control policy models. In: Proceedings of the POLICY, pp 9–16
51. Yu L, France RB, Ray I (2008) Scenario-based static analysis of UML class models. In: Proceedings of the ACM/IEEE 11th international conference on model driven engineering languages and systems (MoDELS), Toulouse, France, pp 234–248
52. Yu L, France RB, Ray I, Sun W (2012) Systematic scenario-based analysis of UML design class models. In: Proceedings of a ICECCS meeting held 18–20 July 2012, Paris, France, pp 86–95