An Efficient Access Control Model for Mobile Ad-Hoc Communities

Sye Loong Keoh and Emil Lupu

Department of Computing, Imperial College, 180, Queen's Gate, London SW7 2AZ
{slk, e.c.lupu}@doc.ic.ac.uk

Abstract. Mobile ad-hoc networks support interactions and collaborations among autonomous devices by enabling users to share resources and provide services to each other, whether collaborations are for business or leisure purposes. It is therefore important to ensure that interactions are subject to authentication and access control in order to restrict access to only those resources and services that the user intends to share. Existing access control models that are based on membership certificates incur redundant verifications and therefore require significant computation. They are inefficient because devices have to repeatedly verify the requestor's certificates and check the authorisation policies for each service access request received. In this paper, we present an efficient access control model that combines a membership list with the role-based access control (RBAC) model. Each ad-hoc network has a coordinator that is responsible for maintaining the membership and broadcasting a signed membership list to all participants at regular intervals. The model authorises a service request if the requestor is listed in the membership list and its assigned role is authorised to perform the requested actions. Through experiments, we have observed the efficiency gains obtained through use of this model.

1 Introduction

The advancement of mobile technology has enabled the establishment of ad-hoc networks which can be formed anytime and anywhere without relying on the availability of a fixed network infrastructure. Ad-hoc networks support *interactions* and *collaborations* among autonomous devices to share resources and services. This type of network is particularly useful for military coalitions and disaster relief operations that require rapid establishment of network connectivity, as well as for business meetings or leisure purposes such as resource sharing.

Security is one of the main concerns in the establishment and management of ad-hoc networks in order to protect the device's resources from misuse and control access to its services. The lack of infrastructure support and *a priori* knowledge between devices introduce difficulties for authentication, membership management, and access control. Typically, a user joins an ad-hoc network because it needs to access resources and services that it does not have as well as to provide various services to others. Hence, users must be able to authenticate

D. Hutter and M. Ullmann (Eds.): SPC 2005, LNCS 3450, pp. 210–224, 2005.

each other in order to control access to their resources and services. Without authentication and access control, collaborations are difficult to establish because users typically do not *trust* any *strangers* to access their resources. However, the need for a device to authenticate each service request incurs redundancy in the verification of credentials. For example, whenever a device requests to join a network, all the existing members have to verify its credentials before they grant the admission permission. As a result, there is a need for an efficient access control model that can reduce redundant computations that will result in delays and consume *power* (a precious commodity for mobile devices).

In this paper, we consider an ad-hoc network as a *community* of autonomous devices that interact and collaborate with each other. Each community has a community specification called *doctrine* [7] that governs the admission of participants and defines a set of authorisation policies that specify their respective privileges. Users and services are represented as roles. Thus, roles are used to group users with the same rights and duties, and also act as a placeholder for common type of services. In the community, admission control is undertaken by a *coordinator* that is responsible for periodically broadcasting a membership list to all participants. The proposed access control model enables service providers to use the membership list and the doctrine to determine the requestor's eligibility to join the community and to decide whether permission can be granted to access the services. Through experiments (c.f. section 7), we observed that this model is more efficient than models relying solely on membership certificates.

In section 2, we present some related work, and outline some of the shortcomings of current access control models for mobile ad-hoc networks. Section 3 presents some background information, while section 4 presents an efficient approach to disseminate a community membership list. We introduce our access control model in section 5 and section 6 describes the implementation. This is followed by a description of an experiment setup and results in section 7. We conclude the paper and highlight future work in section 8.

2 Related Work

In infrastructure-based networks such as the Internet, the use of popular Public-key Infrastructure (PKI) standards such as membership certificates provides a scalable solution to address authentication and access control issues. Membership certificates are issued specifically to a particular user, so that the user only needs to present the certificate to the service provider and then authenticate itself in order to access the service. As a result, various studies [8, 16, 13] have attempted to adapt this solution to ad-hoc networking by setting up dedicated certification authorities (CAs) within the ad-hoc network itself. These authorities are responsible for issuing membership certificates to the participants and authentication is performed by checking these certificates. An approach proposed in [8] selects a group leader to handle the admission control in an ad-hoc group and the leader uses its own public key pair to issue membership certificates to the participants. This approach is not flexible because when the group leader

leaves, all the certificates need to be renewed and the new group leader must re-issue membership certificates to all the participants. To mitigate this problem, the authors introduced the concept of *delegation*, so that delegated group leaders can also issue membership certificates to admit new users. In addition, these delegated leaders are encouraged to issue redundant certificates to existing group members. Consequently, a member could possibly possess more than one certificate issued by various group leaders in the network. This requires more storage for membership certificates, and more computations are needed to issue redundant certificates.

Other similar approaches that adopt threshold cryptography [16, 13] enable multiple mobile nodes in an ad-hoc network to act as CAs, and use a public-key pair for the entire network. The private-key of the network is split into n shares, where $n \geq 3t + 1$ and these key shares are then distributed to n mobile nodes *out-of-band*. Each of the mobile nodes is required to issue a partial certificate share to every new user who wishes to join the network. Thus, a new user has to gather $t + 1$ certificate shares and combine them to form a valid membership certificate. This approach is proposed in order to prevent the single point of failure. However, when $t + 1$ nodes that have a share of the private-key leave the network, these nodes could jointly issue a valid membership certificate. This issue can be mitigated by using proactive secret sharing [6], where key shares are updated periodically in order to prevent collusion, but this requires a great deal of effort and computations to update all the key shares at regular intervals. Furthermore, the additional message exchanges require significant additional power consumption.

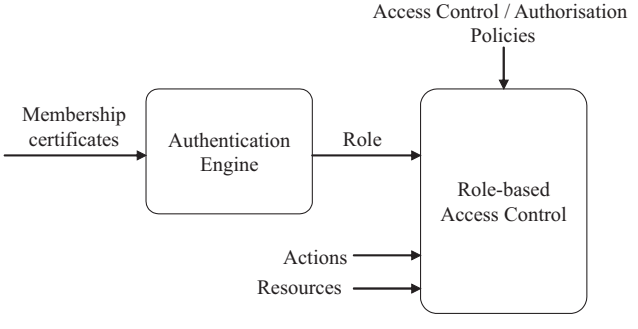

Fig. 1. An access control model that uses the membership certificates in conjunction with RBAC

Figure 1 shows a typical access control model of various approaches that use membership certificates in conjunction with RBAC [12, 3]. For each service request, the requestor must present a valid membership certificate in order to access a service. The authentication engine first authenticates the requestor. Secondly, it checks the membership certificate to determine the participant's

role information, and lastly access is granted to perform the requested actions according to the role information and the authorisation policies.

Models based on the PKI standards exhibit three fundamental issues when deployed in mobile ad-hoc networks. First, they wastes devices' computational resources because for every access request, the service providers have to repeatedly authenticate and verify the membership certificates of the requestor, even if the requestor was previously authenticated. Caching methods can be used, but they introduce additional complexities such as determining the caching period as well as deciding which information can be safely cached. Since mobile devices typically have limited computational resources, this model is heavy weight and not efficient. Second, an ad-hoc network is transient in nature; this implies that certificates are valid only within the lifetime of the ad-hoc network. When the network is dissolved, the certificates become useless since they cannot be used in other ad-hoc networks. As a result, the use of certificates in this approach does not justify the effort of going through the certificate issuance process which is both expensive and laborious. Third, the group leaders or the certification authorities in the ad-hoc network have to periodically broadcast a certificate revocation list (CRL) to all participants, so that they know which participant's membership has been revoked. This also incurs additional computational costs because the authenticity of the CRL must be verified and revocation of certificates needs to be checked periodically as well.

Other solutions that use symmetric key cryptography as a shared group secret-key to prove participants' membership [15, 14, 2] are also less flexible. This is because it is not possible to differentiate one participant from another since all of them know the secret-key. This implies that they all must share the same access privileges if the access permission is granted solely based on the possession of a group secret-key.

3 Background

This section provides some background information in association with the proposed access control model.

3.1 The Community Specification, *Doctrine*

In [7], we have proposed a policy-based approach to establish ad-hoc communities. A community specification called *doctrine* defines the participants in terms of roles, the user-role assignment (URA) policies, the policies governing the behaviour of the participants and the overall constraints of the community. Bootstrapping an instance of an ad-hoc community requires a group of participants to agree to use the same doctrine and to designate a coordinator. The participants must also satisfy the URA policies in order to join the community. The authorisation and obligation policies specified in the doctrine define the access privileges and obligations of the respective roles in the community. The URA policies define the constraints on the attribute certificates that a user must possess in order to be assigned to the requested role in the community. These could

be the user's role in an organisation, position in a company and membership etc. The doctrine also provides the flexibility to specify additional security requirements such as *separation-of-duty*, *cardinality* and *community establishment* constraints. These constraints define the conditions under which a community can be established as well as the conditions to admit a new user.

Typically, a doctrine is issued by an issuer and expressed in XML. It is signed by the issuer, thus ensuring its integrity, and the signature is encoded using XMLSignature [1].

In [7], we have discussed the evolution and management of communities, with emphasis on the community establishment protocols and the underlying trust assumptions. In essence, each community has a *coordinator* that is responsible for handling the admission requests, thus enforcing the URA policies and community constraints, as well as for periodically broadcasting the membership list to all participants. All participants maintain weak consistency of the membership list with the coordinator. This is efficient because broadcast is inexpensive especially in small ad-hoc communities where devices are in proximity of each other. The use of a coordinator is desirable in order to avoid redundant verifications (otherwise each participant would need to verify that all other participants satisfy the URA policies). However, this does introduce an issue of trust in the coordinator. Typically, communities comprise heterogeneous devices with varying computational capabilities ranging from laptops to PDAs, or low-powered sensors. The coordinator must therefore be chosen amongst the devices with the most processing capabilities, e.g., a laptop or PDA. As all the participants must have fulfilled the URA policies in order to join the community, we assume that they trust each other to act as the coordinator. In addition, participants can expect each other to behave according to the policies of the community, which are specified in the doctrine and known to all participants. Lastly, in order to prevent the single point of failure, a new coordinator can be selected when the current one becomes unavailable.

3.2 The TESLA Authenticated Broadcast

Conventional authenticated broadcast requires an asymmetric approach to generate digital signatures. However, TESLA [9] adds a single message authentication code (MAC) to a message for broadcast and achieves an asymmetric digital signature through clock synchronisation and delayed symmetric key disclosure. It therefore makes authentication less expensive. In TESLA authentication, the sender first chooses a random key K_N and generates a one-way key chain by repeatedly applying a one-way hash function H (such as SHA-1) on K_N in order to generate all other keys: $K_{N-1} = \mathrm{H}[K_N]$, $K_{N-2} = \mathrm{H}[K_{N-1}]$, ... In short, $K_i = \mathrm{H}[K_{i+1}] = H^{N-i}[K_N]$. The receiver can therefore easily compute a key K_j from a key K_i provided that j < i using the equation $K_j = H^{i-j}[K_i]$. However, nobody can compute K_i given K_j because of the one-way property of H. A key is considered authentic provided that when the equation is applied, the computed value matches the previously authenticated key on the chain.

Time is divided into uniform time intervals and a key is assigned to each interval. The generated key chain is used in reverse order for the computation of the message's MAC in each time interval. In addition, the sender pre-determines a key disclosure schedule, δ in which it will disclose the key that was used to compute the MAC. For example, in time interval (i + δ), the sender reveals the key K_i. The key disclosure delay, δ must be greater than the round-trip time between the sender and receivers [9]. TESLA requires that the sender and the receivers are loosely time synchronised and each node knows an upper bound on the maximum synchronisation error [10]. This is essential for the receiver to ensure that the message received was computed using a key that has not yet been published. Further details on how to establish loose time synchronisation are discussed in [9].

When a receiver receives a message with a MAC, it must first verify that the key K_i used to compute the MAC has not been disclosed. This is necessary because an adversary could have forged the message if it already knows the disclosed key. Therefore, the security condition imposed in TESLA requires the MAC of the received message to be computed using a key that has not yet been revealed. If this is fulfilled, the message is buffered and the receiver waits for the sender to publish the key. Once the key K_i is published, the receiver authenticates the key and then verifies the MACs of the messages that are stored in the buffer.

4 Dissemination of the Membership List

The membership list is maintained by the coordinator and periodically distributed to all the participants in the community. Thus, all participants maintain weak consistency of the membership list with the coordinator. Typically, the list contains the mappings of all participants' public-keys to their corresponding roles in the community and a membership entry in the list is a tuple of ⟨*node address, node id, public-key, role assignments, time of admission, device capability*⟩. However, using asymmetric cryptography to provide authenticity and integrity of the membership list is expensive. In this section, we present a light-weight approach to disseminate the membership list to participants at regular intervals using the TESLA protocol.

4.1 Upon Bootstrapping of the Community

When a group of users have agreed to use a doctrine to bootstrap a community, the user who initiates the community is selected as the coordinator if it has adequate resources and computational capabilities. The coordinator broadcasts a signed ⟨REQUEST⟩ which includes its credentials and public-key as well as the doctrine to be used in order to bootstrap the community. The coordinator must then ensure that all the replying users satisfy the URA policies and that the community constraints are not violated.

As shown in Figure 2, once the community has been bootstrapped, the coordinator generates a set of TESLA parameters, $\langle K_0, \delta, T_{int}, \ell \rangle$ and broadcasts

Message 1: *The coordinator, co generates a keychain and TESLA parameters which include the duration of a time interval, T_{int} and the key disclosure schedule, δ. Subsequently, co broadcasts a signed message which contains the TESLA parameters and the membership list to all participants.*

co: $K_n \rightarrow \mathrm{H}[K_n]{=}K_{n-1} \rightarrow ... \rightarrow \mathrm{H}[K_1]{=}K_0$
$M_{co} = \langle \mathrm{MEMBERSHIP}, co, CID, TS, K_0, T_{int}, \delta, \ell, \textit{membership list} \rangle$
co $\rightarrow$ *: $\langle M_{co}, \mathrm{Sign}(Priv_{co}, \mathrm{H}[M_{co}]) \rangle$

After T_{int} (1 interval)...

Message 2: *After an interval, co computes a MAC for the updated membership list and broadcasts it to all participants. At the same time, it reveals the secret MAC key according to the key disclosure schedule.*

co: $M_{co} = \langle \mathrm{MEMBERSHIP}, co, TS, CID, I_2, \textit{membership list}, K_{1-\delta} \rangle$
co $\rightarrow$ *: $\langle M_{co}, \mathrm{MAC}(K_1, \mathrm{H}[M_{co}]) \rangle$

*TS = timestamp, CID = community id, ℓ = length of keychain

Fig. 2. The protocol to disseminate the TESLA parameters and the initial membership list

them to all the participants. K_0 is the last generated key in the keychain, δ is the key disclosure delay, T_{int} is the duration of a time interval, and ℓ is the length of the keychain. Together with these parameters, the coordinator includes the initial membership list and then signs them using its private-key. Each community has a community id, CID which is a pseudo-random number generated by the coordinator as an identifier for the community instance. The broadcast is also timestamped. Note that, a digital signature is used at this initial bootstrapping phase because all the participants do not know the TESLA parameters for the community. After a time interval, T_{int}, the coordinator broadcasts the membership list using TESLA. From this point onwards, the coordinator broadcasts the membership list at regular time intervals, i.e., the membership list is included in the key disclosure message according to δ. In addition, all participants must loosely synchonise their clocks with the coordinator's using the simple time synchronisation protocol [9].

Once participants have received the initial membership list, they verify it using the coordinator's public-key. At the same time, they store the TESLA parameters. Subsequent membership list updates are cached by the participants. When the corresponding secret-key, K_i (where i is the interval) is disclosed by the coordinator, the participants check the authenticity of the key and subsequently verify the MAC of the membership lists that are stored in their cache. If successful, they update their own local copy of the membership list.

4.2 When a User Joins or Leaves the Community

When a new user requests to join the community, the coordinator has to first check the URA policies and grant admission to the user if the community constraints are not violated. As illustrated in Figure 3, the coordinator sends a ⟨JOIN REPLY⟩ message to the admitted user. The message consists of the TESLA parameters, $\langle K_0, T_1, T_{int}, \delta, \ell \rangle$ and the membership list. T_1 is an additional parameter and denotes the initial time of interval 1. This enables the new user to compute the current interval with respect to the current time in order to determine the authenticity of a secret MAC key. In addition, the admitted user must loosely synchronise its clock with the coordinator.

Message 1: *The coordinator sends the TESLA parameters, the initial time of interval 1,* T_1 *and the membership list to the newly admitted user,* u_0.

co: $M_{co} = \langle$JOIN REPLY, *co*, u_0, TS, CID, K_0, T_1, T_{int}, δ, ℓ, *membership list*$\rangle$

co $\rightarrow u_0$: $\langle M_{co}, \text{Sign}(Priv_{co}, \text{H}[M_{co}]) \rangle$

Message 2: *After that, the coordinator broadcasts the membership list to all other participants. The membership list is signed using TESLA.*

co: $M_{co} = \langle$MEMBERSHIP, *co*, TS, CID, I_i, *membership list*, $K_{i-\delta} \rangle$

co $\rightarrow$ *: $\langle M_{co}, \text{MAC}(K_i, \text{H}[M_{co}]) \rangle$

Fig. 3. The protocol to disseminate the TESLA parameters to a newly admitted user

After that, the coordinator broadcasts the membership list to all participants and signs the membership list using TESLA. When a participant is detected/considered to have left the community, the coordinator has to update the membership list and broadcasts it to all using the same procedures.

4.3 When a New Coordinator Is Selected

It is possible that the coordinator moves out of range or become unavailable. Since all community participants have a copy of the membership list, they can check whether they are eligible to be the new coordinator. Amongst all the eligible participants, only one participant is selected. The choice is really arbitrary, however for implementation purposes, we usually use the participant with a high CPU capability device who is assigned the *lowest node id* in the membership list as the new coordinator (i.e., the oldest member of the community with a sufficiently powerful device). The new coordinator sends a ⟨RECONSTRUCTION REQUEST⟩ message to all participants. Participants would only respond if the new coordinator is listed in the local copy of their membership lists. This is to prevent malicious users from sending *fake* requests in order to take over the admission control in the community. Existing participants can then rejoin the community, and thus the community can be reconstructed.

Soon after the coordinator has gathered sufficient re-join replies and the community constraints have been satisfied, it generates a new key chain and then broadcasts a new set of TESLA parameters to all participants. The protocol is the same as the bootstrapping protocol as shown previously in section 4.1.

5 The Access Control Model

The access control model exploits the local copy of the membership list that each participant possesses in order to optimise the access control mechanism. Therefore, the service providers only need to check the membership list to determine whether the requestor is a valid member of the community, and do not need to repeatedly verify the requestor's credentials. We also use the RBAC model because the doctrine defines the participants in terms of roles. Hence, permissions are assigned to roles rather than individual identities.

As shown in Figure 4, the access control model consists of a *membership management component* and a *policy enforcement component*. Both of these components run on all users' devices. The *membership management component* is responsible for maintaining the membership list that the participant receives periodically from the coordinator, and verifiying the authenticity and integrity of the list received. In addition, it extracts the authorisation policies from the doctrine that the service providers must enforce and forwards them to various enforcement components. The *policy enforcement component* is responsible for enforcing authorisation policies. As shown in Figure 4, all the service requests must be signed by the requestor, they are intercepted and then the policy enforcement component checks whether permissions can be granted by performing the following:

– **Authenticate the requestor** - The policy enforcement component authenticates the requestor by verifying the signature of the service request. Since

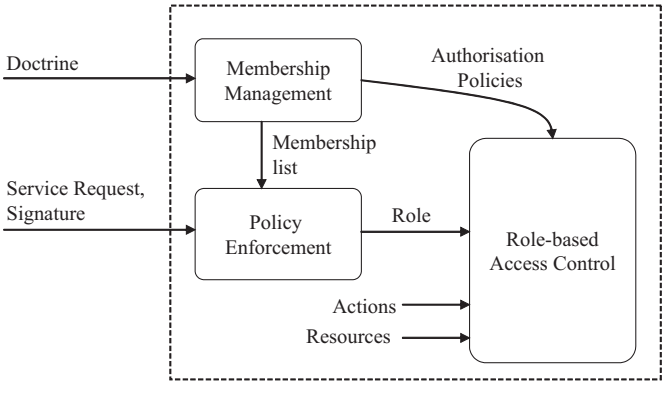

Fig. 4. The proposed access control model that uses the membership list instead of membership certificates

it has access to the membership list, it can obtain the public-key of the requestor in order to verify the signature.
- **Check the requestor's role assignment** - The policy enforcement component determines the role of the requestor by using the membership list.
- **Check permission** - Lastly, based on the role assigned to the requestor and the authorisation policies, a decision is made whether to permit the requested actions or not.

6 Implementation

To implement this model, we have extended the Java security architecture [4], as the default Java security manager and access controller provide a general implementation that can be used to incorporate our access control model.

Figure 5 shows the incorporation of our access control model with the Java security model. Typically, authorisation policies are enforced at the target object. Therefore, when a new community is established using a doctrine, the membership management component extracts the authorisation policies in the doctrine that the participant has to enforce and passes them to the policy enforcement component. Subsequently, the policy enforcement component translates these authorisation policies into Java authorisation policies that can be loaded by the Java security manager.

We have defined a new permission called *ServicePermission* that extends the *java.security.BasicPermission*. The *ServicePermission* represents a permission to access the services, (i.e., invocation of a method on the target object). In addition, the default Java principal-based authorisation only supports the use of X.509 certificates. Therefore, we have extended the *java.security.Principal* to define a new *AdHocPrincipal* that can be used to encapsulate the user's role information in an instance of an ad-hoc community.

Figure 6 shows the translation of an authorisation policy as defined in the doctrine to the corresponding Java authorisation policy. The policy specifies that

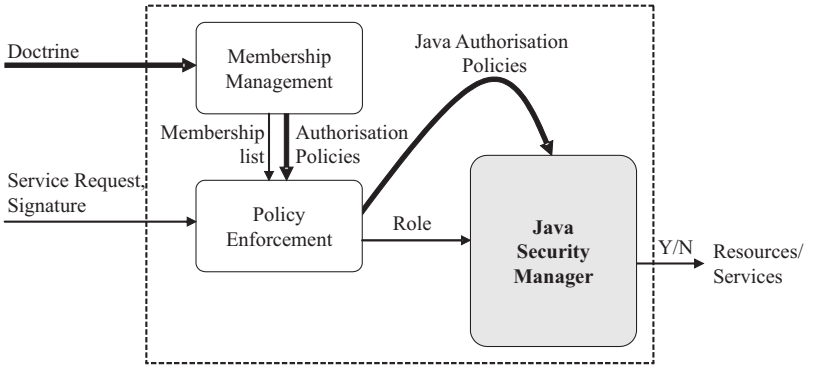

Fig. 5. The incorporation of the proposed access control model with the Java security model

the participants who are assigned to the role *Staff* of the community instance ($CID = 183674$) are granted permission to invoke the method *print* on any target object *Printer*. This policy is enforced by devices which are assigned to the role *Printer* in the community.

```
inst role Staff {
    inst auth+ printAuth {
        Target Printer;
        Action print(); }
}
```

This authorisation policy is being translated into the following Java policy

```
grant Principal AdHocPrincipal "183674-staff" {
    permission ServicePermission "print";
};
```

Fig. 6. This policy defines the rights of participants who are assigned to the role *Staff*. They are authorised to use the print service in the community

In essence, for each access request received, the policy enforcement component determines the requestor's role assignment using the membership list and then it encapsulates the role information in an instance of an *AdHocPrincipal*. Next, it invokes the Java security manager to check whether the requested action is permitted based on the authorisation policies and the role information.

7 Experiment Setup and Results

7.1 Experiments

We developed a proof-of-concept prototype scenario to dynamically establish ad-hoc communities comprising devices for various categories of users such as *Staff, Research Assistants, PhD students, Undergraduate Students*, and intelligent devices such as *Printers*, and *Projectors* in the Department. Users can dynamically bootstrap an ad-hoc community and then use the *print service* and *slide projector service*. These services are implemented using Java RMI invocations in which an authorised participant can invoke the methods *print()* and *project()* on the printer and projector objects.

A doctrine is created to define the roles of the participants as well as their privileges, i.e., the authorisation policies. For example, only *Staff, Research Assistants*, and *PhD students* are authorised to use the *Printer*. This is achieved by first look up the remote object of the *Print Service* and then invoking the *print()* method. The target object, *Printer* checks the eligibility of the requestor and the authorisation policies before it prints the submitted document.

In the experiment, we measured the performance of the policy enforcement component which includes the process of authenticating the requestor, determining its role assignments and invoking the Java security manager to check

the authorisation policies. Performance was measured in terms of *transaction rate*, which is a value provided by *perf4J* [5], a tool for metering and monitoring the performance of Java applications. Essentially, *perf4J* measures the average elapsed time it takes to execute the demarked Java code-units, i.e., a transaction. These operations were executed on PII-350Mhz machines with 128Mb of RAM. We compared the performance of three access control models as follows:

- **Model A (a chain of attribute certificates)** — The requestor has to present a set of attribute certificates that are encoded using Simple Public-Key Infrastructure (SPKI)/Simple Distributed Security Infrastructure (SDSI) [11], the policy enforcement component finds a certificate chain and then checks the URA policies in order to determine the role to be assigned to the requestor. Subsequently, authorisation policies are checked based on the assigned role.
- **Model B (the role membership certificate, RMC)** — The coordinator or the group leader is responsible for issuing RMCs to all participants that satisfy the URA policies. An RMC is encoded using SPKI/SDSI as well. When the requestor requests to access a service, it presents the RMC for verifications. The policy enforcement component verifies the RMC before it grants permission to the requestor.
- **Model C (the membership list)** — All participants maintain weak consistency of a membership list with the coordinator. The requestor does not need to present any certificates, the policy enforcement component checks the membership list and the authorisation policies before granting permission to the requestor.

7.2 Results

Figure 7 illustrates the comparison of performance measurements of various access control models. For all models, the elapsed time decreases (i.e., the transaction rate increases) as invocations become more frequent, e.g., when there are 60 service invocations in a time period of 30 minutes (i.e., an invocation every 30 seconds), compared to 10 service invocations. Note that there is a significant performance gap between Model C and other two models; the transaction rate for Model C is at least two times higher for any number of invocations. More specifically, the average transaction rate for Model C is of 19 and 26 when the corresponding number of invocations are 30 and 60. These figures are approximately four times higher than Model B and nine times higher than Model A. In other words, when using the membership list, the *Print Service* can handle up to 25.6 invocations per second as compared to Model B that uses role membership certificates with less than 6.5 transactions per second.

Thus, the proposed model requires significantly less computations than the other two. Model A appears to require the most processing time because it has to perform complex operations to evaluate the certificate chain, while Model B also requires the execution of public-key cryptography which is expensive.

We have also measured the overhead incurred when verifying the authenticity of the membership list received at regular intervals. The average transaction rate

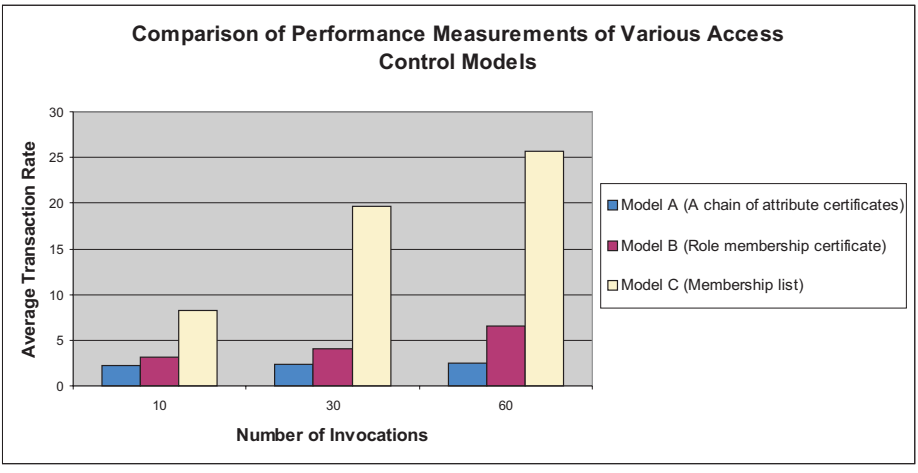

Fig. 7. Performance measurements of the access controller for various access control models

Table 1. The size of the membership messages, |ML| = size of the membership list

Message Content	size
⟨MEMBERSHIP⟩ with TESLA parameters	2.133kB + |ML|
⟨MEMBERSHIP⟩ with a secret key	1.417kB + |ML|

to verify a TESLA key is ≈ 66.06, while the rate to authenticate the MAC of the membership list is approximately 56.63. A variant of Model C which uses digital signatures instead of TESLA can be used to provide authenticity of the membership list. In this alternative model, we observed that the average transaction rate for verifying digital signatures is ≈ 46.10. This indicates that for a powerful machine e.g., PII-350 Mhz, the use of digital signatures does not have major impact on the performance as it is slightly heavier than the use of TESLA.

In addition, as shown in Table 1, we have measured the approximate sizes of the ⟨MEMBERSHIP⟩ messages to convey TESLA parameters and to reveal a 128-bit secret-key.

8 Conclusion and Future Work

This paper has presented an access control model that requires significantly less computations than the other models, and is therefore more efficient. Although there is a small amount of overhead incurred by the periodic broadcast of membership list using TESLA, this overhead is not significant because TESLA uses symmetric-key cryptography and broadcast in wireless networks is inexpensive. In all other computations, our access control model does not involve additional cryptographic operations except for verifying the signature of the service requests. However, this verification is required in all models encountered. This

indicates that the overhead imposed by the periodic broadcast of the community membership list is justifiable for small to medium-size communities as the performance gains in terms of computation are significant by comparison with models where service providers have to re-verify requestor's credentials or certificates for each access request. In the latter models, we have also not taken into account the required computations to verify the certificate revocation lists (CRLs).

However, a consequence of the use of TESLA is that only weak consistency between the membership of the community as perceived by different participants, can be guaranteed. Slight inconsistencies of the membership list among the participants are inevitable because when a user joins or leaves the community at time t_i of interval j, the coordinator needs to broadcast the updated membership list to all participants, but the membership list will only be authenticated and therefore valid after $\lambda = (\delta * T_{int}) + (ET_j - t_i)$, where δ is the key disclosure schedule, T_{int} is the duration of a time interval and ET_j is the end time of interval j. This means that the participants have to wait untill the secret-key used to sign the membership list is published by the coordinator in order to update their local membership list. As a result, for the time period λ, the membership lists of the coordinator and the newly admitted user are in sync, while the other participants still hold the previously authenticated membership list. This can be mitigated by minimising δ, as this also minimises the time period of having inconsistent membership lists, thereby shortening the waiting time for which the newly admitted user can start interacting with other participants. However, this implies that the membership list is broadcast more frequently thereby increasing the resources (computation and transmission power) required. An alternative approach to avoid this drawback is to use digital signatures to sign the membership list. However, a reasonably powerful device is required to perform digital signature verifications, while constrained devices might not be capable of continuously executing this operation.

One advantage of using the doctrine to express the underlying security policies is that all participants are aware of the admission criteria and their respective privileges in an ad-hoc community. Hence, this builds *trust* among the participants and ensures that the behaviour of the coordinator and the other participants can be monitored. Consequently, immediate actions can be taken if the coordinator is found to have misbehaved. However, we are still investigating the use of the presented model in conjunction with a monitoring framework that is able to detect policy violations.

Finally, the approach to broadcast a membership list is suitable for relatively small to medium mobile ad-hoc networks that typically interconnect devices in the vicinity of each other. The use of a membership list is two-fold, first it serves as an input to the policy enforcement component to determine the eligibility of the requestor before authorising its actions. Second, it can be used as the CRLs of the community. On this basis, the proposed access control model is efficient and effective although only weak consistency of the membership list can be maintained.

Acknowledgements

We gratefully acknowledge financial support from the EPSRC for AEDUS research grant GR/R95715/01 and from the EU FP6 TrustCOM Project No. 01945. In addition, we are indebted for many comments and suggestions to our colleagues Dr. Naranker Dulay and Prof. Morris Sloman.

References

1. M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing, 2002.
2. J. Bray and C. Sturman. *Bluetooth Connect Without Cables*. Prentice Hall PTR, 2000.
3. D. Ferraiolo and R. Kuhn. Role-Based Access Controls. In *Proceedings of the 15th National Computer Security Conference*, pages 554 – 563. NIST, 1992.
4. L. Gong. *Inside Java 2 Platform Security Architecture, API Design and Implementation*. Addison-Wesley, 1999.
5. J. Hebert. The Perf4J API, 2002.
6. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing Or: How to Cope with Perpetual Leakage. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 339–352. Springer-Verlag, 1995.
7. S.L. Keoh, E. Lupu, and M. Sloman. *PEACE* : A Policy-based Establishment of Ad-hoc Communities. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC), Tucson, Arizona, USA*. IEEE Computer Society, December 2004.
8. S. Mäki, T. Aura, and M. Hietalahti. Robust Membership Management for Ad-hoc Groups. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems (NORSEC 2000), Reykjavik, Iceland*, 2000.
9. A. Perrig, R. Canetti, J.D. Tygar, and D. Song. The Tesla Broadcast Authentication Protocol. *RSA Cryptobytes*, 2002.
10. A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Mobile Computing and Networking*, pages 189–199, 2001.
11. R.L. Rivest and B. Lampson. SDSI – A Simple Distributed Security Infrastructure. Presented at CRYPTO'96 Rumpsession, 1996.
12. R.S. Sandhu and E.J. Coyne. Role-Based Access Control Models. *IEEE Computer*, 29(8):38–47, 1996.
13. N. Saxena, G. Tsudik, and J.H. Yi. Admission Control in Peer-to-Peer: Design and Performance Evaluation. In *Proceedings of the First ACM Workshop on Security of Ad-hoc and Sensor Networks (SASN), Fairfax, Virginia, USA.*, October 2003.
14. F. Stajano. The Resurrecting Duckling – What Next? In *Proceedings of the 8th International Workshop on Security Protocols*, LCNS. Springer-Verlag, 2000.
15. F. Stajano and R.J. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proceedings of the 7th International Workshop on Security Protocols*, LCNS. Springer-Verlag, 1999.
16. L. Zhou and Z. J. Haas. Securing Ad-Hoc Networks. *IEEE Network Magazine*, 13(6), November/December 1999.