An Access Control Architecture for Distributing Trust in Pervasive Computing Environments

Raquel Hill  
School of Informatics and Computing  
Indiana University  
Bloomington, Indiana  
ralhill@indiana.edu

Jalal Al-Muhtadi  
College of Computer and Information Sciences  
King Saud University  
Riyadh, Saudi Arabia  
jalal@ksu.edu.sa

William E. Byrd  
School of Informatics and Computing  
Indiana University  
Bloomington, Indiana  
webyrd@indiana.edu

Abstract—Pervasive computing infrastructure is highly distributed and it is essential to develop security mechanisms that enhance the security of the system by distributing trust among the various infrastructure components. We present a novel access control architecture explicitly designed to distribute trust that combines threshold cryptography, multi-layer encryption, and mediated access to contextual data to support dynamically changing access control permissions. We present several models of our access control infrastructure and evaluate how well each design distributes trust and limits the behavior of misbehaving components. We also simulate the behavior of our threshold-based access control scheme and evaluate the overhead of each infrastructure model.

I. INTRODUCTION

Current pervasive computing environments encompass sensors, actuators and specialized components for capturing and processing many types of contextual information. “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves” [1]. This contextual information is then used as a default computational parameter in the system. The various components that are responsible for capturing context information are often distributed and autonomous. In many scenarios, there are situations where access to information need to be constrained to well-defined, high-level, activities that are taking place. For example, in a smart medical environment a doctor may need to consult with other specialists about a patient, such that the infrastructure does not allow the viewing of the patient’s records except when the supervising doctor and the authorized specialists are located in the proper location, and no unauthorized personnel are located in the same space. In a military scenario, sensitive information regarding battle plans should not be allowed to be displayed in the smart command and control room, unless all (or a subset) of the authorized senior personnel are present, the doors are locked, and no unauthorized personnel are present. In such scenarios, high-level activities (e.g., a closed meeting taking place in a specific room, etc.) can be implied by fusing sensors or gathering raw data from various sensors, and deriving higher-level contextual information. For example, if the environment is able to detect the presence of several people, who are sitting at a large table in a room and talking in an orderly fashion, then this could imply that a meeting is taking place. When restricting access to information until a higher-level context is realized, two challenges are present. First, the low-level sensed data is coming from a plethora of sensors, therefore, relying on a single component to fuse the data and derive higher-level context is putting trust into a single component rather than distributing the trust. Second, in many cases, it may not be possible to sense context accurately. For example, it may be difficult to precisely sense the location of an object or accurately determine the activity in the room. There is, thus, an element of uncertainty in many sensed contexts. This is often caused by misplaced sensors or because of a sensor “blind spot” in a room. The access control system must be able to cope with uncertainty or missing data.

To overcome the aforementioned challenges, we extend our proposed framework with threshold cryptography support. Threshold cryptography (TC) involves sharing of a key by multiple entities (called shareholders) who are engaged in encryption or decryption. The goal is to distribute trust among a number of entities instead of trusting a single entity. We offer that security mechanisms used within the proper context can ensure confidentiality policies as well as meet requirements for efficiency. In this paper we incorporate threshold cryptography into the context-based encryption mechanism presented in [2]. We use threshold cryptography to provide a secure mechanism for enforcing role-based access control policies whose permissions have been further constrained by contextual information. Our novel use of threshold cryptography enables us to leverage the dynamic nature of context to support flexible access control policies and to distribute trust among components within the ubiquitous computing infrastructure.

We make the following contributions:
• First, we use threshold cryptography to distribute trust within a pervasive computing environment.
• Second, we present several models of our access control infrastructure and evaluate how well each design distributes trust and limits the behavior of misbehaving components.
• Third, we simulate the behavior of our threshold-based access control scheme and evaluate the overhead of each model.

This paper is structured as follows. Section II presents related work, while section III describes the architecture of our proposed system. Section IV analyzes the security of our system, while section V discusses our prototype implementation. Finally, section VI concludes and discusses future work.

II. RELATED WORK

A Role-Based Access Control (RBAC) [3], [4] bases access control decisions on the functions a user is allowed to perform within an organization; these functions are modeled as roles, the key concept in RBAC. Each role is associated with a set of permissions, which are its rights on objects, and a set of users, which are members of the role and get the permissions. The users cannot pass access permissions on to other users at their discretion.

While RBAC was designed to meet the needs of industry and civilian government, once specified, RBAC policies are static and cannot meet the access control needs of a dynamic, distributed pervasive computing environment. In pervasive computing environments, contextual changes may trigger a change in a user’s access permissions. Therefore, one specific challenge to providing location and context aware security services is the efficient integration of contextual triggers into the authorization mechanism. Current location-based and context-based access control research extends role-based access control (RBAC) systems by enabling users to model and specify spatial or contextual constraints, and the concept of roles is extended to deal with context information [5]–[15]. Hansen et al. define locations based on organizational location infrastructure, which leads to a logical representation of location that reflects the organizational location infrastructure and the organizational security policies [5], [7]. Bertino et al. [6] present Geo-RBAC, which uses spatial entities to model objects, user positions, and geographically bounded roles. Roles within a Geo-RBAC system are activated based on the position of the user. Geo-RBAC also uses logical and device independent positions, such as road, town or region to present the location of the user. In [8], RBAC is extended to support context-aware access control for collaborative tasks. The work introduces an integrated access control model that is aware of the context associated with an ongoing activity, and utilizes the current context in providing dynamic access control. Bertino et al [10] extends role based access control to support temporal constraints on the enabling/disabling of roles. Temporal dependencies are expressed by means of active rules that are automatically executed when the specified actions occur. These temporal dependencies can also be used to constrain the set of roles that a particular user can activate at a given time instant. The firing of a rule or role trigger may cause a role to be enabled/disabled either immediately, or after an explicitly specified amount of time. Bhatti et al. [9] extend RBAC with trust levels, context awareness, and temporal restrictions. The Aware Home project has extended RBAC with object and environment roles [14] that are used to define context-aware security policies such as those based on temporal authorizations. However, they do not address permissions under specific high-level contextual situations. Kumar et al. [16] also consider incorporating context into the RBAC model with contexts and context filters. dRBAC [17] is a decentralized trust-management and access control mechanism for systems spanning multiple administrative domains. Unlike other approaches that use contextual information to extend, RBAC Sampemane et al. [18] define an access control model for Active Spaces that introduces system, space and application roles and uses RBAC to specify the user-role and role-permission assignments. Sampemane et al indirectly support contextual changes through the use of sessions, which determine the amount of time that permissions remain active. A session ends when an event occurs within the Active Space that changes the operating context; for example, a user enters or leaves the space or a new application is started. We use Sampemane’s model to specify access control policies for our use case scenario.

A. Encryption-based Access Control

Several recent works [2], [19]–[22] use encryption as an access control mechanism. Fu [20] uses encryption as an access control mechanism to allow protected content to be stored on untrusted servers. Data is encrypted using a content key. The content key is encrypted using the group key and all members of a group use their group key to access the content key and decrypt the protected content. Group membership is dynamic in that members may join and leave the group at anytime. Key regression is used to rekey during times when membership has changed. The main contribution of this work is that access control is decentralized and data access is not mediated by the content publisher, which makes the system scale. Fu rekeys when a member leaves a group to prevent future access. Our approach differs in that a user who has left the group cannot access the protected content, although he or she still possesses the group key. We deny access without distributing new group keys.

Goyal et al [21] define encryption based access control system that associates an attribute based policy with an encryption key and cipher texts. If the attributes of the key and a piece of cipher texts matches, then the key can be used
to decrypt the data. Goyal’s approach does not use a secret sharing or threshold cryptography scheme because they do not want entities to work together to bypass an access control policy. Our work differs from Goyal et al in several ways. First, our goal is to allow coarse-grain access control, therefore allowing authorized users full access to a restricted data object. Furthermore, we use threshold cryptography to distribute trust among components within our access control infrastructure. We foresee using Goyal’s approach to restrict the portions of an object that may be accessed by a user or group of users, while still using threshold cryptography to enforce contextual policies and limit the effects of compromised infrastructure components.

Payne [22] presents a vault security model to protect against a privileged attacker. Each user has a separate secure repository in which they may store confidential data. This data includes cryptographic keys used by applications and one-way hash values of sensitive system programs which are subsequently verified on access. Also stored are credentials that are used to access cryptographically protected files. User vaults are decrypted upon login after the user supplies the necessary decryption key. Files within the vault are accessible only to cryptographically verified processes for the remainder of a user’s session. Cryptographically verifying the integrity of the programs that are able to access the contents of a user’s vault ensures that a privileged attacker cannot subvert the security of the scheme by modifying trusted code. Payne limits the effects of a compromised node by restricting it’s access to protect resources. Our approach differs in that we limit the effects of a compromised node by distributing trust among multiple infrastructure, thereby limiting the scope of the compromised node’s influence.

Harrington et al [19] present a distributed access control framework for a networked file system that uses encryption as an access control mechanism. Files are encrypted using a symmetric key algorithm to provide confidentiality. Asymmetric cryptography is used to generate and verify digital signatures. Our work differs in that we use a multi-layer encryption scheme where the outer layer of encryption is associated with a context-based policy and is removed by the access control infrastructure.

Our context-aware access control scheme extends the context-based encryption scheme that is presented in [2]. In [2] encryption is used to restrict access to data resources. Access to resources is limited to users within a specific geographic region. Location data is used to determine whether data should be decrypted or not. The decrypted data is then given to users whose location has been verified.

III. ARCHITECTURE

Threshold cryptography (TC) involves sharing of a key by multiple entities (called shareholders) who are engaged in encryption or decryption. The goal is to distribute trust among a number of entities instead of trusting a single entity. Additionally, this method allows the system to tolerate node failures to a certain extent. Threshold schemes generally involve key generation, encryption, share generation, share verification, and share combining algorithms. Share generation, for data confidentiality and integrity, is the basic requirement of any TC scheme. Threshold models can be broadly divided into single-secret sharing threshold like Shamir’s k-out-of-n scheme based on Lagrange’s interpolation [23], threshold-sharing functions like geometric based threshold [24], and threshold signature schemes as in [25], [26]. These schemes implement threshold variants of RSA cryptographic algorithms. Shoup’s [25], [26] signature scheme is unforgeable assuming that the RSA problem is hard; it is non-interactive, thus it does not require synchronous communications; and it bounds the size of the signature share by the size of the RSA modulus times some small constant. Given the distributed nature of our environment, we use Shoup’s threshold signature scheme to distribute trust among sensors within our active space environment.

We present our architectural overview in Figure 1. Our access control system consists of several components, including sensors, a contextual service, policy service and an event service. We describe each component below.

- Sensors - Sensors include any device that provides information about the environment, including for example, door locks, authentication mechanisms, temperature, lighting levels, sound levels, time and date, schedule, patient vital signs, etc.
- Context Service - The context service captures and processes contextual information from various sensors and deduces higher-level context. The context service consists of two components, sensor brokers and combiners. Sensor brokers mediate access to data produced by sensors and provide primitives for enabling communication with other components and services in a smart space infrastructure. For lightweight sensors, sensor brokers are simply a dedicated PC on which the sensor’s component runs. Sensor brokers store the key shares of the corresponding sensors. Once a sensor broker receives data from it corresponding sensor, the sensor broker creates a context message and a partial signature for that message using its key share. This message is then forwarded to the combiner.
- The combiner combines signature shares to create signed context messages. When the combiner receives k signature shares from sensor brokers that correspond to the same message, the combiner combines the signature shares to create a signed message. This message is forwarded by the combiner to the policy service. Note that sensor brokers do not forward their key shares to the combiner, thereby protecting those shares from a possibly compromised combiner.
Policy Service - The policy service derives the higher-level context that corresponds to a layer of encryption. The policy service contains two components, policy enforcement servers and reasoning engines. The policy enforcement servers verify signatures on messages that it receives from combiners and store the corresponding information contextual information. The servers pass this information along to the reasoning engine once a user request is received and an access control decision is needed.

The design of the reasoning engine is based on the ideas outlined by Ranganathan et al. [27], [28]. The reasoning engine uses first-order logic to reason about contextual information that it receives from the policy enforcement servers.

Event Service - The event service that allows events to be communicated between distributed objects. The event service utilizes a publish-subscribe model for dispersing events. With the event service, users can create secure event channels where channel participants are restricted to authorized entities and sensitive events are encrypted, as described in [29]. All relevant sensor components within the smart space infrastructure are connected through a special secure event channel. The event channel is not explicitly depicted in the architectural overview, but consider any communications channel to be a part of the overall event system.

Data within the Active Space is stored in encrypted format. We use the layered encryption approach proposed in [2]. The outermost layer of encryption corresponds to a context-based key, while the inner layer of encryption corresponds to a group-based key. The context-based encryption layer is removed by the infrastructure if the policy services determines that the context has been met.

A. Assumptions and Notation

We make several assumptions when designing the overall system. First we assume that a trusted Certificate Authority (CA) exists, and that key shares, are securely generated and pre-loaded on brokers. We assume that doubly-encrypted data has been loaded onto the data store and that network communications are reliable. Finally, we assume that sensors are relatively static and that they do not join and leave the Active Space at will.

B. System Architecture

We now examine two variants of our proposed architecture.

Figure 1 shows what we call the rekeying variant of our proposed architecture (this name will become clear shortly when we discuss the non-rekeying variant). The figure shows the steps the system executes to authenticate a user, and to fetch and partially-decrypt a file for the user.

When the user enters an area, she may be authenticated by one or more physical sensors (e.g., a fingerprint reader) (step 1). Each sensor is connected to a sensor broker, which processes the sensor’s data, and, if the user is authenticated by the sensor, the corresponding broker (partially) signs an authentication message using the sensor broker’s key share. This authentication message, complete with partial signature
(or signature share) is then forwarded to the combiner server (step 2). Once combiner has received authentication messages and signature shares from enough distinct sensor brokers, the combiner attempts to combine the signature shares to form a traditional digital signature for the user authentication message, which it then sends to server 1 (step 3). Server 1 then verifies the combined digital signature, ensuring that a sufficient number of sensor brokers have authenticated the user (step 4). Server 1 forwards this authentication message to Server 2, which also verifies the digital signature (step 5).

The authenticated user then asks for a resource, such as a business document. The request is relayed to server 1, which checks with the context policy service to ensure the user should have access to the resource. Server 1 then sends the requested resource to server 2 (step 4, revisited). All resources saved in the data store are doubly-encrypted: once with the user’s group key, and once with server 2’s context key.

Server 2 independently checks with the context policy service to ensure the user should have access to the resource. If so, server 2 removes a single layer of encryption using its context key (step 5, revisited), and forwards the singly-encrypted data to the user (step 6). The user then removes the final layer of encryption using her group key (step 7).

Figure 2 presents an alternative to the rekeying variant presented in Figure 1. This variant is identical to the rekeying variant, with one exception: the doubly-encrypted requested resource is partially-decrypted by server 1, and partially-decrypted by server 2. The user receives a fully-decrypted file, via the (encrypted) data channel.

The advantage of this variant is that, since decryption is via the context keys on servers 1 and 2 rather than the user’s group key, it is not necessary to distribute new group keys when someone leaves a group; hence we call this variant the non-rekeying architecture. However, the non-rekeying variant requires greater trust in server 2, which fully decrypts the requested resource before forwarding it to the user via the encrypted data channel. Since the primary interest of this paper is trust distribution, we consider the rekeying variant to be canonical—however, the non-rekeying variant might be a better choice if users are frequently changing groups.

The message protocol figure, Figure 3, shows every message sent in the following simple scenario: a single user enters a meeting room, authenticates with two of three authentication sensors (numeric keypads, fingerprint scanners, RFID readers, etc.), then requests a business document. The flow of information through the sensor brokers, combiner, servers 1 and 2, and the data channel should be familiar from Figures 1 and 2. New to this figure is the nonce generator, which is responsible for generating a new nonce whenever someone opens the door to the conference room. The new nonce is broadcast to the servers and brokers, which in turn delete any user-related state, thereby forcing users to reauthenticate in order to access protected resources. The use of the new nonce in subsequent messages protects against replay attacks.

Figure 3 also shows the control channel, which acts as an intermediary between user requests and servers 1 and 2. The control channel also forwards authentication confirmation messages to the user from server 2, and is responsible for telling the user to reauthenticate when a new nonce is broadcast by the nonce generator.

### IV. Security Analysis

We distribute trust at multiple layers within our architecture. Trust is distributed among sensors in that multiple sensors are required to produce a piece of contextual information. The benefits of this scheme is that trust is distributed rather than trusting a single entity for making decisions on whether the higher-level context is realized or not. In addition, it allows the system to cope with some level of uncertainty if some sensors are unable to give accurate readings. In addition, a hacker would have to compromise at least k sensor brokers before being able to forge contextual information and possibly violate a confidentiality policy. Trust is also distributed among the policy service and users in that the policy service removes the layer of encryption that is associated with the context-based policy and users.
remove the inner layer of encryption that corresponds to user/group permissions. This distribution of trust among the policy service and users prevents compromised infrastructure components from gaining unauthorized access to restricted resources.

Our proposed architecture provides for efficient key management. Sensor brokers are assigned one key share per event or meeting held within the Active Space. Since sensor brokers compute partial signatures and do reveal their key shares to the combiner, the combiner cannot forge contextual messages. Therefore, sensor brokers may use the same key shares throughout the duration of an event.

V. IMPLEMENTATION AND EVALUATION

In this section we discuss our prototype implementation of the rekeying and non-rekeying trust distribution architectures presented in section III-B, including the prototype’s features, limitations, and performance.

The main purpose of our prototype was to ensure our proposed architecture is viable, and to refine the details of the message protocol presented in Figure 3. Implementing the prototype accomplished both of these goals.

The core of our prototype is written in Ikarus Scheme version 0.0.4-rc1+, an R6RS Scheme [30] compiler that produces native Intel code. We chose Scheme for its flexibility and abstraction capabilities, which, for example, made it trivial to write the reasoning engine of section V-A. Ikarus extends R6RS Scheme with support for TCP/IP networking, spawning and managing system processes, and a C-compatible Foreign Function Interface.

To generate and verify signature shares we use the existing “threshsig” Java library [31], which implements Shoup’s practical threshold signatures [25]. Our prototype communicates with the threshsig library via wrapper code written in Scheme and Java.

All messages and data files are encrypted using the Advanced Encryption Standard (AES) using the cipher-block chaining (CBC) mode and 128-bit keys. The prototype uses OpenSSL version 1.0.0-beta3 to perform this encryption, and to produce and verify message signatures. Digital signatures are produced using 2048-bit RSA keys.

Our prototype models each entity in Figure 3 (for example, users and sensor brokers) as a separate process. All communication between processes is via TCP/IP.

A. Reasoning Engine

As mentioned in section III, the policy service relies on a reasoning engine that uses first-order logic to reason about contextual information; this reasoning engine determines which resources a user can access, and when, depending on the user’s context.

For our prototype we implemented a simple reasoning engine in the style of a Lisp interpreter. The engine accepts contextual policies in a declarative language that can express

\[
\text{event} \ \text{followup-restructuring-mtg}
\begin{align*}
&\{ \text{all} \\
&\{ \text{location} \ \text{conference-room ceo} \} \\
&\{ \text{location} \ \text{conference-room executive-assistant} \} \\
&\{ \text{at-least} \ 3 \\
&\{ \text{location} \ \text{conference-room engineering-vp} \} \\
&\{ \text{location} \ \text{conference-room sales-vp} \} \\
&\{ \text{location} \ \text{conference-room marketing-vp} \} \\
&\{ \text{location} \ \text{conference-room human-resources-vp} \} \\
&\{ \text{location} \ \text{conference-room finance-vp} \} \\
&\{ \text{after} \ \{ \text{date} \ 2009 \ \text{april} \ 7 \} \ \{ \text{time} \ 8 \ 00 \} \} \\
&\{ \text{before} \ \{ \text{date} \ 2009 \ \text{april} \ 7 \} \ \{ \text{time} \ 9 \ 30 \} \} \}
\end{align*}
\]

\[\text{Figure 4.} \ \text{Example declarative context description for an executive meeting.}\]

start and end times, locations, and roles. For example, Figure 4 shows a contextual policy that is satisfied if the CEO and executive assistant of a company, along with at least three of five vice presidents, and in the conference room between 8:00 a.m. and 9:30 a.m. on April 7, 2009.

We intend to build a more sophisticated language extending the language presented by Ranganathan and Campbell [28], along with a context reasoning engine that provides the user with detailed information on why access to a resource was denied.

B. Simplifications and Limitations

To simplify the implementation of our prototype, we store all public, private, and symmetric keys in a single folder. A full implementation of our architecture might use a Certificate Authority to manage public keys, and OpenSSL to perform symmetric key exchange. Alternatively, keys could be pre-loaded onto brokers and servers if the environment is mostly static.

Another simplification is that sensors are represented as software processes; a full implementation might include door sensors, fingerprint scanners, RFID tag readers, and other physical sensors.

Our proposed architecture architecture assumes the existence of encrypted data channels that entities can subscribe to or publish to, as described in [29]. While the prototype does encrypt all network traffic, it does not implement a publish-and-subscribe mechanism; as a result, the same message may need to be sent individually to multiple recipients. The most important limitation of the prototype is that the combiner only verifies the sensor brokers’ signature shares, rather than combining them into a normal RSA signature. Although the threshsig library makes it easy to generate and verify signature shares, threshsig does not support creation of complete signatures from those shares. As a result, our prototype verifies the signature shares, then signs the authentication message with the combiner’s signature. We intend to extend the threshsig library to allow creation of combined signatures; we anticipate that this extension will be straightforward, and will have negligible impact on the performance of the overall system.
C. Performance

Our prototype was designed for ease of testing and flexibility rather than speed; still, it may be illuminating to examine a few simple benchmarks. The performance data presented in this section should therefore be considered worst case.

Tables I and II present performance data for a simple use case. Here a single user enters a room, authenticates with two out of three sensors, then requests a file. After receiving the file, the door is opened, requiring the user to reauthenticate. Table I presents data for the rekeying architecture, in which requested resource data is decrypted by Server 2 and by the user. Table II presents data for the non-rekeying architecture, in which requested resource data is decrypted by servers 1 and 2 rather than by the user.

Each table displays the number of encrypted bytes sent over the network, as a result of requesting files of different sizes. The original, unencrypted file size is given in the File Size column, followed by the size of the same file after being encrypted twice using 128-bit AES in CBC mode. Bytes Sent (Resource Data) specifies the number of encrypted bytes representing the file that are sent over the network; this number is roughly twice the size of the encrypted file, since encrypted data is passed between servers 1 and 2 before being passed to the user. Bytes Sent (Message Data) specifies the number of encrypted bytes comprising the messages in Figure 3, excluding the resource data.

Also presented are the roundtrip times it took for the user to be authenticated (Time (auth)), from the time the sensor detects the user to the time the user receives the authentication confirmation message. Finally, each table contains the time it took for the user to receive resource data after requesting it. All times are in seconds; due to uninteresting technical reasons, the precision of the timer was limited to 1 second.

During each test exactly 16 messages were sent. The number of messages sent in an implementation supporting publish-and-subscribe might be less.

The tables show that the performance of the rekeying and non-rekeying approaches is essentially indistinguishable. This is to be expected, since the messages sent by the two schemes are identical, other than the number of layers of encryption applied to the payloads of the resource messages.

We benchmarked the reasoning engine described in section V-A separately from the rest of the system. The engine imposes negligible overhead, requiring less than a millisecond to reason about several simple meeting scenarios.

VI. Conclusion & Future Work

In this paper, we present a novel framework that distributes trust among infrastructure components within a pervasive computing environment and enables context-aware access control via the use of encryption and threshold cryptography. The approach is novel in that it combines the use of TC and heterogeneous high-level contexts to make an access control decision. Our simulations show that our multilayered access control mechanisms can operate efficiently even for complex scenarios and increasing key sizes. We have also illustrated how our threshold cryptography enabled access control mechanism can be used to enforce policies that have multiple contextual conditions. We envision this mechanism being used in situations to increase the confidence in sensor readings by combining the output of multiple sensors via the use of threshold cryptography. We foresee context being used in conjunction with identity and group membership to provide finer-grain access control services.

REFERENCES


