A Secure Context Distribution Framework for Peer-based Pervasive Systems

Wolfgang Apolinarski, Marcus Handte, Pedro José Marrón
Networked Embedded Systems Group
University of Duisburg-Essen and Fraunhofer IAIS
Germany
{firstname.lastname}@uni-due.de

***Abstract* — Pervasive computing envisions seamless and distraction-free application support for everyday user tasks. Achieving this requires a high degree of automation. In many scenarios, the basis for automation is context information that can be acquired unobtrusively by means of sensors. Consequently, it is vital to ensure the validity of the context information, especially, in cases where automatic decisions can have severe security implications. In smart environments, the validity of context information can be ensured simply using a centralized context storage that is securely connected to all trusted sensors. In peer-based systems such a centralized approach cannot be applied. Instead, it is necessary to use all devices to distribute context information which requires additional precautions to ensure its validity. In this paper, we derive the requirements on secure context distribution for peer-based systems. Furthermore, we describe a generic distribution framework to enable the usage of context information in security critical applications. On the basis of a prototypical implementation, we present an evaluation indicating that the proposed framework can achieve a high level of security while being applicable to many scenarios.**

Keywords-component; context information, peer-based, secure distribution, validation, pervasive computing

I. INTRODUCTION

Pervasive computing envisions seamless support for everyday user tasks by means of devices that are integrated in the user's environment. Due to their integration, many devices are specialized and resource-poor and due to user and device mobility, the resulting pervasive systems are typically dynamic. As a consequence, many pervasive applications are inherently distributed since a single device alone cannot provide thorough task support. Combined with the dynamics, this creates execution environments that demand a high degree of configurability and adaptability.

In addition to seamlessness, pervasive computing also strives for distraction-free task support. Thus, it is usually not feasible to shift the responsibility of adapting an application to the user. Instead, providing the desired user experience requires the application developer to strike the right balance between manual control and automation. However, the benefit of automation can quickly be nullified by inappropriate decisions. This is especially problemantic if the automated decisions may have security implications, e.g. if they may compromise the user's privacy.

To avoid inappropriate decisions, it is often necessary to consider a large number of variables. Besides from technical characteristics of the execution environment such as the available devices and services, a significant set of variables is usually bound to the state of the physical world which is commonly referred to as context information or simply context. Context may entail the location of objects and devices, for example, and it is typically supposed to be gathered unobtrusively by means of embedded sensors. Consequently, it is necessary to ensure the validity of context information that is used for automation.

To ensure the validity, existing systems usually rely on a trusted infrastructure that consists of a centralized context service with permanent secure connections to all relevant sensors. This approach can be applied easily to smart environments since they are often built around a centralized server that manages a single administrative domain. In contrast to that, peer-based systems are typically fully distributed since they cannot rely on the permanent availability of any device. Moreover, they may span devices from several administrative domains which can make it impossible to define a single trustworthy context service.

As a consequence, it is necessary to use the available devices to distribute context information in peer-based systems. However, this approach requires additional precautions to ensure the validity of the context information. In this paper, we derive the requirements on secure context distribution in peer-based systems. Furthermore, we describe a generic distribution framework to enable the usage of context information in security critical applications. On the basis of a prototypical implementation, we present an evaluation indicating that the proposed framework can achieve a security level that is comparable to a centralized system while being applicable to a broad range of scenarios.

The remainder of this paper is structured as follows. In Section II, we introduce our basic system model. In Section III, we provide an example scenario. Thereafter, in Section IV, we derive the resulting requirements. In Section V, we introduce our generic framework to enable the secure distribution and in Section VI, we present an evaluation on the basis of a prototypical implementation. Finally, Section VII describes related work and Section VIII concludes the paper with a summary and an outlook on future work.

II. SYSTEM MODEL

As presented in [1] and [6], our work focuses on peer-based pervasive systems. In these systems, devices that are within communication range connect to each other on-the-fly

using short-range wireless communication such as Bluetooth or WLAN. Due to miniaturization and specialization, the devices encountered in these systems are often resource-poor. As a result, they need to interact with each other in order to provide thorough support for user tasks. Due to mobility, the available set of devices is continuously fluctuating. Thus, in contrast to smart environments, peer-based systems cannot make use of a central coordinator. Instead, they must coordinate in a decentralized manner.

In order to reduce the amount of manual user inputs, the systems rely on context information. Usually, this context information is gathered unobtrusively by means of sensors that are integrated into some of the devices. As a simplification, we assume that the information that is gathered by the sensors is representing information about some of the devices in their vicinity. Given that the sensing range is often smaller than the communication range, we argue that this simplification is not overly restricting.

From a security perspective, the devices that are connected to each other at a particular point in time may span multiple domains that are administered independently. As a result, it is not safe to assume that all devices are equally trustworthy. Instead, the devices that are part of one administration domain may use their resources to change the behavior of the devices from another maliciously. As a consequence, it is necessary to ensure that critical decisions are based solely on valid context from trustworthy devices.

III. Example Scenario

To clarify the system model, consider the example scenario shown in Figure 1 that depicts a research campus similar to the institute center that hosts Fraunhofer IAIS. To limit the access to the campus to legitimate persons, the whole area is enclosed by a fence and both, employees and visitors need to pass by a gatekeeper at the main entrance. Due to the size of the campus, there is a significant distance between the gatekeeper's location and the office buildings which makes it impossible to keep track of the visitors after they passed the gate. As a consequence, employees may have to pick up first-time visitors at the main entrance to ensure that they are not getting lost. In addition, the sheer size of the campus also makes it hard to ensure that illegitimate persons are not simply climbing over a fence.

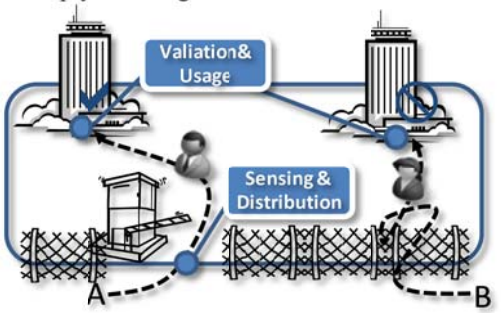

Figure 1. Example Scenario

To improve this situation, the access control performed by the gatekeeper can be revalidated at the entrance of the individual buildings by means of a pervasive application. When a person enters through the main gate, a sensor recognizes this and generates a corresponding piece of context information. Since the gatekeeper's office is not continuously connected to all building entrances, the sensor stores the context information on the mobile phone of the person. Once the person arrives at a building entrance, an actuator that is mounted to the entrance requests the context information from the mobile phone and it solely unlocks the entrance, if the person has been detected by the sensor at the gatekeeper (cf. A) which prevents illegitimate access (cf. B).

IV. Requirements

From the peer-based system model presented previously and the desire to support the utilization of context information to automate security critical decisions, we can derive the requirements for context distribution as follows:

Decentralized provisioning: As peer-based systems cannot provide guarantees about the availability of devices, the context information must be made available in a decentralized manner. More specifically, to ensure support for arbitrary disconnections, a particular piece of information must be stored on all devices to which it relates to.

Generic mechanisms: The mechanisms used for context distribution should be generic with respect to the context information that shall be distributed. This ensures that they are applicable to a broad range of different applications.

Reliable security: In order to enable the use of context information for security critical decisions, it is necessary to ensure that the information has actually been generated by a reliable source (authenticity). Furthermore, it is necessary to ensure that the information cannot be altered when stored on a device that is not trustworthy (integrity). Finally, in some cases it may be necessary to ensure that the information has been generated recently (freshness).

Configurable trust: Since peer-based systems may span multiple administrative domains, it is not viable to rely on the context information generated from arbitrary devices. Instead, it is necessary to allow the configuration of trust in order to support the secure utilization of context generated in different domains.

Low resource usage: Last but not least, in order to be applicable to resource-poor devices, the provisioning of the context information should not be a resource intensive task. Instead, the mechanisms for distribution should exhibit low resource utilization without endangering the security.

V. Framework

In the following, we describe our framework to support the secure distribution of context information. To do this, we first provide an overview of the basic framework. Thereafter, we describe the individual mechanisms and protocols. Then, we describe some extensions to reduce the resource utilization. Finally, we briefly outline the prototypical implementation which we evaluate in the next section.

A. Overview

To support the decentralized provisioning of context information while providing strong security, our framework

decouples the tasks of sensing and distributing context from the task of validating it in space and time. To do this, the basic version of our framework relies on asymmetric cryptography. As we will show in the evaluation, this approach is suitable for devices with limited resources such as Sunspots. However, to reduce the resource requirements even further, we provide a symmetric alternative.

As basis for asymmetric cryptography, we require that each device belonging to the same administrative domain is equipped with a key pair and a certificate that is signed by a common root certificate. Thus, the root certificate represents a single administrative domain and it can be used to identify the domain's trusted devices. Furthermore, we require that devices are identified by the fingerprint of their certificate which ensures that the identification is hard to compromise. The generation and distribution of the keys and certificates is done in an offline step that can be performed using standard tools such as Openssl.

Figure 2. Framework Overview

On top of this setup, the basic variant of the framework differentiates three functionalities. As depicted in Figure 2, all functionalities rely on a key store that holds the keys and certificates described previously. The generator functionality is responsible for perceiving its environment by means of some sensor and for distributing the resulting context information. The storage functionality stores the context information provided by generators for later usage. The validator functionality retrieves and validates context information and depending on the result of the validation, it may initiate an action, e.g. by means of an actuator.

To perform the validation, the validator functionality must determine whether the context has been issued by a trusted generator component. To this, the framework introduces a distribution protocol between the generator and the storage and a validation protocol between the storage and the validator. In the following, we provide a detailed description of these protocols. Thereafter, we describe an extension of the framework that replaces the distribution protocol with a symmetric variant.

B. *Distribution Protocol*

When the sensing unit of a generator perceives some new piece of information about a device, the generator stores the information in the storage of this device. To do this, the generator and the storage execute the distribution protocol. As depicted in Figure 3, the distribution protocol consists of a single update message that is initiated by the generator.

Besides the type of the context (CTP) and the actual information (CONTEXT), the update also contains the identifiers of the generator device (FP_G) and the targeted storage device (FP_S). The context type and the actual information can be arbitrary byte sequences. The type itself is later on used in the validation protocol to request a particular piece of context. As stated previously, the identifiers refer to the fingerprints of the corresponding certificates. Thus, they provide a strong associating between the generator and the storage. As we explain later on, this ensures that a piece of context information that has been generated for one device cannot be used by another. Finally, the update also contains a timestamp (T_G) of the generator as well as its certificate ($CERT_G$). During the validation, the timestamp is used to ensure the freshness of the information. The certificate is needed to enable the validation of arbitrary generators without knowing them a priori.

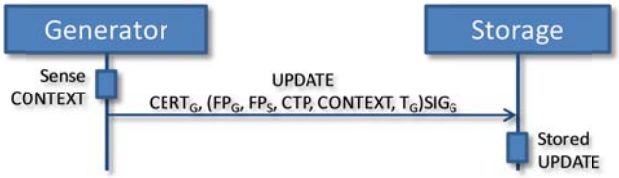

Figure 3. Distribution Protocol

To ensure the authenticity and integrity of the update, the identifiers as well as the context type, information and time are signed using the private key of the generator. If necessary, this also allows the storage device to determine whether the context has been issued by a trustworthy generator. To do this, storage would have to verify that $CERT_G$ has been signed by a trusted root certificate and that SIG_G is valid. However, this validation step at the storage is not required to enforce the security goals introduced earlier. Instead, the storage can simply store all updates as received.

C. *Validation Protocol*

If the validation is required, the validator is responsible for retrieving the context. As depicted in Figure 4, this requires a request and a response that (1) authenticates the storage and (2) transmits the verifiable information.

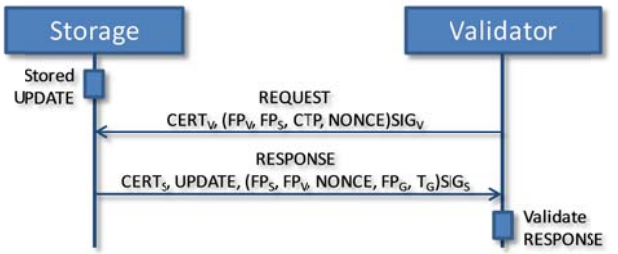

Figure 4. Validation Protocol

To achieve this, the request is composed of the context type (CTP) that shall be retrieved, the identifiers of the validator (FP_V) and the storage (FP_S), a nonce (NONCE) that is generated by the validator as well as its certificate ($CERT_V$). The identifiers, context type and the nonce are signed by the validator which can be used to enable the storage device to restrict requests to trusted validators. To do this, the storage verifies that $CERT_V$ is signed by a trusted root and that SIG_V is valid. However, just like the validation of the generator of an update, this is an optional step.

Similarly, the response to this request is composed of the certificate of the identifiers FP_V and FP_S and the nonce that was included in the request. Furthermore, it includes the identifier of the generator (FP_G) and the timestamp of the update (T_G) as well as the certificate of the storage ($CERT_S$) and the original update message. Finally, the message includes a signature for the identifiers, the nonce and the timestamp that is generated by the storage.

In order to validate the response, the validator first determines whether the nonces are identical. Thereafter, it determines whether the CTP in the update is as requested. Then, it determines whether FP_S, FP_G and T_G in the response are identical to the values in the update and whether the fingerprint of $CERT_S$ and $CERT_G$ are matching FP_S and FP_G. Finally, the validator verifies that the certificate of the generator is signed by a trusted root and that the signatures in the update (SIG_G) and the response (SIG_V) are valid.

Once this validation process has succeeded, the validator can accept the context. Thereby the process ensures that the storage device is authentic since it must be equipped with the private key for $CERT_S$ to create SIG_S and the response cannot be a replay as it includes FP_V and the correct nonce. Furthermore, the update message has not been changed during the transmission since the signature SIG_V also spans FP_G and T_G and they are identical to the update. Finally, the update has been issued for the storage device since it includes FP_S and it can be trusted since it $CERT_G$ is signed by a trusted root and thus, the generator can be trusted.

In order to determine whether the context information itself is fresh (enough), the validator can use the timestamp of the generator (T_G). However, this requires that the clocks of the validator and the generator are (loosely) synchronized. This can either be achieved by an external time source such as a UTC or GPS receiver. Alternatively, trusted storage devices that regularly pass the generator and the validators can also perform the synchronization. To do this in a secure manner, it is possible to reuse the protocols described above. Thereby, the trusted storage device acts as a generator for the time and the actual context generator acts as a validator. To set the time on the context generator, the trusted storage device simply generates an update containing the current time that is used within the validation procedure performed by the generator.

D. Extensions

As indicated in the framework overview, it is possible to further reduce the resource utilization of the framework for generators by replacing the asymmetric signatures with their symmetric counterparts. For this to work, however, the validators and the generators must share the same symmetric key. If this can be achieved by means of key distribution, the protocols described above can be applied directly by removing $CERT_G$ from the update and by replacing SIG_G with a symmetric signature based on the shared key.

However, besides increasing the effort for key distribution, the main drawback of this approach is a loss in flexibility when context shall be used across different administrative domains. In the asymmetric case, it is possible to define unidirectional trust-relations. To do this, a validator can simply be configured to trust the generators of a certain (set of) domain(s). In the symmetric case, this is not possible as the distribution of the key would result in a symmetric trust-relationship since every validator could forge the values of the generator.

To mitigate this problem, our framework introduces so-called generator bridges or simply bridges. The bridges are responsible for replacing the symmetric signatures with asymmetric ones. To do this, they request the context information from the storage, they validate the symmetric signature and they create a corresponding update message. This requires them to be configured with the symmetric key of the generator and an asymmetric key pair that shall represent the generator.

E. Implementation

To evaluate the framework, we have implemented it as an extension to BASE [1], our communication middleware for peer-based pervasive systems. BASE is implemented in Java (on top of J2ME CLDC) and provides basic middleware services for spontaneous interaction such as device discovery and local as well as remote communication. In addition, BASE provides a light-weight service abstraction that we used to implement the framework functionalities.

We have implemented individual services for symmetric and asymmetric generators, storages, validators and bridges. These services provide interfaces to distribute arbitrary context information that can be used in different applications. Towards this end, the generator services must be extended with a sensing unit that creates the context information. The validators must be extended with an actuation unit that requests the context information. In addition to these services, we have implemented a service to distribute the root certificates within an administration domain. Although not being secure, in general, this service enables the user to detect new root certificates and to decide whether they want to trust them. In addition, it could be used to distribute certificate revocation lists, however, our current prototype does not support this.

All services share a common key store functionality which is used to configure the device and its trust-relations. Thus, besides from storing the certificate and key of a device, the key store also stores trustworthy and un-trusted (but discovered) certificates. To represent these certificates, we rely on the X.509 standard. In order to associate the fingerprints of the certificates with devices, we use them as BASE's system identifier.

To realize the cryptographic algorithms, we reuse the implementations of the J2ME version of Bouncycastle library [2]. This allows the utilization of RSA and ECC as asymmetric methods. Both can be used at the same time by different generators or administrative domains. For symmetric authentication, we rely on HMAC [8] using the SHA-1 hash algorithm. Of course, other types could easily be added, so our framework does not depend on a particular cryptographic algorithm.

Since Bouncycastle is applicable to all devices that provide a Java virtual machine with J2ME CLDC support, the library itself is not optimized for a particular type of

device. Thus, to get realistic estimations for the overhead of the framework, we are basing our evaluation on an optimized implementation for Sunspots. This implementation makes use of the Sunspot SSL library which provides a fast 160-bits ECC implementation using SECP160r1. The resulting encryption strength is comparable to RSA1024. However, it is noteworthy that the optimized version of our prototype can interact with non-optimized versions without modification.

VI. EVALUATION

In the following, we evaluate the framework with respect to the requirements identified in Section IV. We first discuss the qualitative characteristics. Thereafter, we provide a set of benchmarks to quantify the resource utilization.

A. Discussion

Due to the fact that the update messages explicitly identify the source and the target of the context information by means of fingerprints, there is a strong association between the generator and the storage. By authenticating the storage and the context information during validation, some other storage cannot illegitimately use the context information. As a consequence, our framework supports the utilization of devices that are not trustworthy per se and thus, it allows decentralized operation.

Our validation framework is generic as it can be used to distribute any type of context since the services do not make assumptions on the data representation. In our current implementation, the services use byte sequences with equality matching. Yet, the integration of more complex type systems and matching operators would be straight forward.

With respect to reliable security, the signatures ensure that the context information cannot be altered and that a storage device must poses the appropriate private key to use the context. While this prevents attacks such as copying or modifying the context information, the presented framework cannot stop a device from sharing its private key which makes devices indistinguishable. For example in the scenario introduced in Section III, a visitor that legitimately passed the gatekeeper could share the context information with an intruder that jumps over the fence. However, if the visitor and the intruder are cooperating, the visitor could also simply use his storage device to open the door for the intruder. Thus, it is not possible to prevent such attacks technically by solely using cryptography in general. The second possible attack that is not prevented by the framework is a man-in-the-middle in the validation protocol. Instead of responding directly to the request message, an intruder could simply forward it to the legitimate storage that holds the context. Once the legitimate device responds, the intruder then forwards the response to the validator. When the message arrives at the validator, the validation succeeds since the validator actually validates the legitimate device. Yet, for this attack to work, the intruder must mask as the legitimate device (e.g. by copying the fingerprint) and it must be connected to both, the legitimate device and the validator device, simultaneously. From a scenario perspective, this is similar to an intruder that is slipping through the door that has been opened by a legitimate user. Thus, it is possible to complicate such attacks by reducing the communication range. From a protocol perspective, it is possible to initiate the validation on the storage device or to have the user accept an incoming request message manually. As a result, the attack would no longer be possible or could be detected at the price of an increased level of manual interaction.

With respect to configurable trust, the framework enables validators to freely model trust on the devices of different administrative domains by means of a configuration of the key store. Due to the use of asymmetric cryptography, it is possible to model unidirectional relationships as well. This is especially useful in the context of business environments where the devices of an employee may trust the sensors of the company but not vice versa. In addition to the not-trusted and trusted categorization performed by our current prototype, it would be straight-forward to integrate a less coarse-grained notion of trust by introducing detailed classification of root certificates in the key store.

B. Measurements

In order to quantify the resource utilization of the framework, we installed our prototypical implementation on Sunspot devices (RED SDK) and we computed a series of benchmarks shown in Table 1. To get meaningful numbers, we repeated each measurement 20000 times for HMAC and 200 times for ECC. Table 1 shows the mean and the 95% confidence interval of the repetitions.

TABLE I. COMPUTATION TIME IN MILLISECONS

Mechanism	*Mean (95% Confidence Interval)*
HMAC	11.34 (8.98, 13.69)
ECC (Signature)	644.57 (473.14, 816.00)
ECC (Validation)	796.07 (607.12, 985.01)

To distribute some piece of context information, a generator must create a single signature in addition to transmitting the context information which results in an additional overhead of 644.57 ms. By using the symmetric mechanism, this overhead can be even further reduced to 11.34 ms. If the storage wants to perform the optional validation of the signature, it needs to perform 2 validations which increase the total delay by 1592.14 ms.

To use a previously distributed piece of context information, a validator must first generate the signed request and then it needs to perform one validation of the storage and two validations of the contained update which results in a total overhead of 3032.78 ms. In addition to that, the storage must also create a signature for the response which corresponds to 644.57 ms. If a storage wants to validate the request, this introduces another 2 validations.

As a consequence, the total overhead for context distribution and usage boils down to 7506.2 ms, if the storage performs all validations and 4321.92 ms, if the storage simply accepts all updates and requests. Clearly, this overhead makes it impossible to use the framework for context information that is changing at a high rate. However, if the context exhibits this behavior, it is likely that the context generator and the context consumer are directly

connected. Thus, it is easier to ensure the validity of context by securing the connection. In cases, where direct connections are not possible, the additional overhead is not unreasonably high. As a consequence, we argue that this approach is applicable to a broad range of scenarios.

VII. Related Work

Most other comparable approaches to utilize context information are based on a central server that is trustworthy. As a consequence, such approaches are not applicable to peer-based systems. In [5], Al-Muhtadi et al. present a context based security suite for the GAIA environment. There a central server saves all data encrypted with a key based on the context (e.g. the location). Accessing the encrypted data will be possible, if the central server can verify the needed context information. GAIA uses a centralized data storage and access control. All the encryption that the mobile devices are performing is based on symmetric group keys. Furthermore, the context based data will be en- and decrypted by the central server, who also determines the current context of all devices.

The security in the virtual home environment [3] is also focused on a central authority that allows securing the communication and enables access control. All requests are routed to a central server which decides if the requesting device is allowed to perform the request. In contrast to this approach, ours allows the data to be distributed through the whole network, so every device carries the data while preserving integrity. Also the verification process is distributed and every device in the network can validate context information.

In addition to centralized approaches, some authors also tried to use context information based security mechanisms in a de-centralized fashion. Robinson et al. [9] creates a shared secret which depends on the room acoustics. So every device in a room should have the same key since they are time-synchronized and therefore creating their key at the same time. This idea depends on the assumption that the room acoustics is almost the same, independent from the place where the device is located. While this could be the case in some rooms, not all rooms fulfill this special condition. They suggest a periodical re-keying, to re-enable the communication with devices which calculated a different key. This key can only be used to secure communication in a room, not to allow secured communication within one company. Our solution enables the verification of context information by every device in a domain which can span an arbitrary big area. In addition, our framework also allows inter-domain context validation, if desired.

Kagal et al. [7] distributes trust to devices of a foreign domain by using the personnel that is working in the domain as room managers. These managers can grant other persons or devices the same rights that they currently own in this room. So a manager could allow a guest device to use the office printer that he is also allowed to use. Personal trust is used as a substitute for secure context information. Also the room managers are responsible for granting their access rights to someone else. Our approach does not depend on personal trust as is not superior in comparison with secured information, especially when considering social engineering.

VIII. Conclusion

Achieving the vision of pervasive computing requires the usage of context information for automation. Especially in cases where automated decision may have security implications, ensuring the validity of context information is unavoidable. In this paper, we derived the requirements on secure context distribution and usage in peer-based systems. Furthermore, we described a generic framework to satisfy them. Our evaluation suggests that the framework can achieve a high level of security that keeps up with current Internet standards while being applicable to many scenarios.

At the present time, we are integrating the presented context distribution framework into our generic role assignment system [6] as part of the PECES European project. This will enable the secure distribution of roles to dynamically form smart spaces and to enable their interaction across insecure networks such as the Internet.

Acknowledgment

This work has been partially supported by CONET (Cooperating Objects Network of Excellence) and PECES (PErvasive Computing in Embedded Systems), both funded by the European Commission under FP7 with contract numbers FP7-2007-2-224053 and FP7-224342-ICT-2007-2.

References

[1] C. Becker, G. Schiele, H. Gubbels, K. Rothermel, “BASE - A Micro-broker-based Middleware For Pervasive Computing”, First IEEE International Conference on Pervasive Computing and Communications (PerCom 03), pp. 443-451, March 23-26, Fort Worth, USA, 2003

[2] “Bouncy Castle Java lightweight cryptography API”, http://www.bouncycastle.org, version 1.44 , released on the 6th October 2009

[3] U. Büker, M. Kuehle „Virtual Home Environment: Security Architecture for Ticket Based Services“, Proceedings of ITEA VHE Workshop, Shaker Verlag, Aachen, 2002

[4] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, “RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, May 2006

[5] J. Al-Muhtadi, R. Hill, R. Campbell and M. D. Mickunas, „Context and Location-Aware Encryption for Pervasive Computing Environments“, Fourth IEEE International Conference on Pervasive Computing and Communications Workshops, (PERCOMW'06), pages 283-289, 2006

[6] M. Haroon, M. Handte, P. Marrón, “Generic Role Assignment: A Uniform Middleware Abstraction for Configuration of Pervasive Systems”, PerWare Workshop at the Seventh Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2009), March 2009.

[7] L. Kagal, T. Finin, A. Joshi, „Moving from Security to Distributed Trust in Ubiquitous Computing Environments“, IEEE Computers, December 2001

[8] H. Krawczyk, M. Bellare, R. Canetti, “RFC 2104: HMAC: Keyed-Hashing for Message Authentication”, February 1997

[9] P. Robinson, M. Beigl, „Trust Context Spaces: An Infrastructure for Pervasive Security in Context-Aware Environments“, Lecture Notes in Computer Science, Security in Pervasive Computing, pages 157-172, First International Conference, Boppard, Germany, March 12-14, 2003