Auctions for Secure Multi-Party Policy Negotiation in Ambient Intelligence

Julian Schütte  
Fraunhofer Institute SIT  
Email: julian.schuette@sit.fraunhofer.de

Stephan Heuser  
Fraunhofer Institute SIT  
Email: stephan.heuser@sit.fraunhofer.de

Abstract—The advent of distributed and ad-hoc-connected systems such as in Ambient Intelligence applications confronts developers with the question on how to specify QoS- and security policies beforehand, without knowing the exact capabilities and requirements of the platforms which will be present at run time. Especially when peers need to adhere to an appropriate trade-off between security and performance, on-the-fly negotiation protocols are required to allow peers to autonomously agree on a common policy. In this paper we present a framework for secure multi-party decision protocols based on auctions. Besides a lightweight implementation, our main focus was on integrating security features for avoiding manipulation of the agreement and reducing the attractiveness of denial of service attacks. As a prototype, we have realized two different auction protocols on top of our framework and used them in a mobile collaboration scenario using a set of Android devices. Experiments with the prototype confirm that auctions are a promising approach for multi-party negotiation of common policies and that the proposed framework facilitates their secure integration into existing applications.

I. INTRODUCTION

With ongoing miniaturization and increasing availability of wireless communication protocols on mobile and resource-restricted platforms, applications come closer to the vision of Ambient Intelligence (AmI). In such applications, devices like sensors, smartphones or controllers co-operate with each other in order to create a user experience of an “intelligent” and context-aware environment that reacts on changing situations and delivers context-adapted information and services. One of the main challenges in this area is to establish a common level of security across heterogeneous devices and services which are not known at design time, but are rather discovered at run time based on their properties. While in static setups, developers can beforehand choose appropriate security mechanisms and configurations which are suited to run on all involved platforms and match the developer’s security requirements, this is not possible in dynamic and ad-hoc-connected AmI environments. It is therefore desirable to make the system self-protective in the way that developers only specify their requirements at design time and the system automatically resolves and applies appropriate mechanisms at run time. This requires policy languages for specifying and protocols for negotiating a trade-off that considers requirements and capabilities of the different involved platforms and takes the user’s preferences into account.

Many protocols such as WS-SecurityPolicy [1] or SSL allow to agree on common security mechanisms to be used in a client/server setting. Yet, these protocol do neither support a multi-party negotiation, nor do they take different preferences of the endpoints into account. Rather, the server simply provides a set of acceptable mechanisms and the client chooses among them.

A promising approach to design more powerful negotiation protocols is to apply ideas from microeconomics and mechanism design [2]. Mechanism design is concerned with creating systems based on game theoretic procedures in order to achieve an overall goal (in our case, finding an optimal common policy) by modelling multiple players who try to optimize their personal welfare. Such procedures can be voting- or auction protocols, for example, and their application for decision-making protocols in distributed systems has gained momentum recently [3]. In this paper, we concentrate on auction protocols, as they are more generic than votings and less work has been done before on using them for policy negotiations. Our goal was to design a development framework that would allow for rapid prototyping of a secure auction-based policy negotiation in AmI applications. The framework includes a set of exemplary auction protocols (an all-pay auction and variants of a Vickrey auction), but can be easily extended by creating new protocols in a model-driven way. The purpose of the proposed framework is to facilitate the creation of distributed “self-adapting” and “self-protecting” applications, which are able to automatically achieve an optimal trade-off between the preferences of different agents, such as between security, bandwidth and CPU consumption, for example.

The rest of this paper is structured as follows: in the following section II we motivate the need for multi-party negotiation by introducing an example scenario and provide an overview of the two auction types which are included in the current framework prototype (further auction types can be easily added). In section III we review work related to ours. Then, in section IV we introduce the software architecture of our framework and explain the functionality of the different components. Section V presents the prototype implementation and discusses its strengths and weaknesses. Section VI, finally, concludes the paper and sketches future work.
II. Motivation

At first, we motivate our work by giving a brief introductory scenario and an overview of auction mechanisms.

A. Scenario

A collaboration platform using mobile devices, like smartphones, netbooks or tablet PCs, allows its users to utilize several cooperation tools, like video- and audio-conferencing and cooperative editing of files. Since a huge variety of devices is available, it is reasonable to assume that not all these devices provide the same features. For example, some phones may lack a 3.5G modem and can only communicate using 3G, which provides significantly less bandwidth. In this case, a mechanism has to be devised for the devices to agree on a maximum bandwidth that may be used by the cooperation platform. Especially if used in a business environment, security becomes an issue: properties like message authenticity and confidentiality are desirable when dealing with business-critical data. However, not every operating system for mobile devices supports the same set of cryptographic protocols. A negotiation procedure is thus needed to specify a common security policy. User preferences may have an influence on the negotiation as well. Suppose one of the devices is low on battery power, so its user wants to disallow video conferencing and security in order to save power. Such user preferences must be taken into account during the negotiation process.

B. Auctions

An established approach to distributed consensus are voting protocols. Recently, auction-based approaches have been discussed in the computer science community [3]. Auction-based consensus relies on a market model in which the bidders bid on the goods being sold. In the context of this paper, different policies represent the goods the bidders bid on the goods being sold. In the context of this paper, different policies represent the goods the bidders place bids on. We will now describe two auction protocols with desirable properties for the use in an AmI-Scenario.

1) Vickrey Auction: A Vickrey auction [4] is a so-called second-price sealed-bid auction. Each bidder sends one final confidential bid (thus “sealed bid”) on the good for sale to the auction host. The bidder which has sent the highest bid wins the auction and has to pay a price for his bid which is as high as the second highest bid (thus “second price”). In this case, the dominant strategy, i.e., the strategy which optimizes the individual outcome of a player and is thus assumed to be followed by each bidder, is to sent a bid amounting exactly to the value the bidders ascribes to the good [4]. The bidders do not gain any advantage by sending higher or lower bids, since this behavior has no predictable positive influence on the price they have to pay or the likelihood that they will win the auction. However, Vickrey auctions are susceptible to attacks based on untruthful auctioneers and bidder collusion [5].

2) All-pay Auction: An all-pay auction bears some similarity to voting procedures. Like in the case of the Vickrey auction each participant provides one sealed bid to the auction host for the good. However, each participant has to pay a price for each of his bids. For the calculation of the prices several different approaches are feasible, like a calculation depending on the behavior of the respective bidder or on the result of the auction. Furthermore, alternative approaches to determining the result of the auction, like a social choice rule [6], may be used.

III. State of the Art

WS-Policy [7] and WS-SecurityPolicy [1] are two specifications which describe a language to formulate (security) policies and a process to establish a common policy between two parties in a client-server scenario. Yet, dealing with multiple parties is not covered by these specifications. Recently, the idea of applying approaches from mechanism design for distributed decision-making has gained increased attention [2], [3].

Auctions describe a market mechanism which may be used for negotiating policies between multiple parties. Common problems solved using auction-based approaches are scheduling and resource-management problems [8], [9]. In [10] and [11] the authors propose using auctions combined with semantic representations of preferences in order to identify those web services in a market which do optimally match the preferences of a client. Capra et al. used auctions for distributed consensus in a middleware designed for AmI applications [12]. While they were able to show that auctions are well suited for the negotiation of policies in AmI scenarios, only one auction protocol was specified and implemented. Furthermore, security considerations for the negotiation of policies have not been examined.

In [13] the author presents a auction protocol for uniform auctions based on the ElGamal homomorphic encryption scheme. The protocol does not require an auction host who has to be trusted by all auction peers. However, the framework described in this paper should allow a developer to implement different auction protocols and compare the results.

FIPA-compliant multi-agent-systems, such as Jade [14], provide an environment for autonomous agents and communication between them. While an agent-based approach is the underlying principle of many negotiation protocols we argue that the implementation of the rather comprehensive FIPA-specifications is not well suited for mobile, resource-constrained devices.

IV. Architecture

As shown in the previous sections, there is a need for more powerful and secure negotiation protocols if features like self-protection are to be achieved in distributed systems. In this section, we present the architecture of the framework...
we designed for supporting secure auction-based policy negotiations.

A. Overview

The framework we propose is based on a component- and service-oriented architecture. The framework architecture is designed to cope with the constraints of mobile devices. Since distributed consensus among the peers of a network is to be reached, each framework instance needs to be able to communicate with other instances running on other peers. As we are using a service-based platform it is reasonable to use this approach for inter-peer communication as well. For the implementation of our prototype we used the OSGi platform [15] which provides these features, yet the framework architecture is independent from OSGi.

B. Components of the framework

In this section, we will describe the major components of the framework and their interactions. Figure 1 depicts the components and their dependencies at the bidder’s and at the auctioneer’s side. In general, there is no need to explicitly distinguish between bidders and auctioneers as all components can be deployed on each endpoint and the bidder or auctioneer role can be chosen at run time.

1) Service Discovery: Since the framework is designed for use in a distributed environment some mechanism is needed to find services running on the local framework instance as well as remote services running on framework instances located on different peers. Additionally, service discovery has to be performed in a secure way. Peers need to be able to authenticate service announcements as well as the services themselves. Authentication of the service announcements is realized using public key cryptography. Service announcements are signed by the service announcement entity. The public key corresponding to the private key used to sign the announcements has to be available to all peers in form of a certificate. All peers need to be able to verify the identity of the service announcement entity using this certificate.

2) Remote-Access: Once a service has been discovered, framework components need a way to make use of the functionality it provides. From a software developers point of view, access to remote services is not fundamentally different than access to local services. While service discovery handles identifying illegitimate service announcements, authentication of the services themselves is done using the Remote Access component and certificates. The service endpoint URI is contained within the certificate’s common name field. The root and intermediate certificates required to build a valid certificate chain need to be available on the peer’s platform.

Services are accessible using interfaces describing the operations they provide. Since the framework is designed for resource-constrained devices, accessing remote services must not introduce unnecessary overhead regarding resources, such as memory, battery life and network bandwidth usage.

3) Logging: To simplify tracing errors during the implementation of different algorithms for distributed policy negotiation debugging and logging is needed. While the framework relies on the remote debugging facilities provided by the target platform SDK, logging is done using a corresponding component.

4) Cryptography: One of the top priorities of this framework is to provide methods to implement secure policy negotiation. Cryptographic operations provide the developer with the ability to ensure confidentiality, integrity and authenticity of messages exchanged between peers. The cryptography component provides functions such as key storage and generation, asymmetric and symmetric encryption, certificate and signature validation, remote service authenticity validation and secret sharing to other components. All cryptographic algorithms (except secret sharing) are configurable and exchangeable without having to modify the source code.

Furthermore, auctions are based on a market model which uses a form of virtual money that is exchanged between auctioneer and bidders. Consequently, messages representing an amount of money have to be protected against manipulation. Virtual money is a representation of a certain amount and is generated by the auctioneer, which subsequently provides it to the auction peers. Our framework uses encryption to ensure confidentiality of all messages exchanged between auction host and the auction peers, which holds for money distribution messages as well. Furthermore, an HMAC is used to provide authenticity and integrity of the distributed amounts, thereby ensuring that no bidder is able to generate money on his own. The key $K_{hmac}$ is used to generate these HMACs and is generated by the auctioneer upon startup. Additionally, whenever virtual money is distributed at time $t$ a new random number $rand_t$ is generated and used as an input to the HMAC algorithm. Upon each of these payment operations, the auction host checks the validity of all amounts using $K_{hmac}$. The symmetric HMAC key may be distributed among the bidders using a $t/n$ secret sharing scheme [16], [17]. Distributing the key among $n$ peers allows a party of $t$ peers to restore the key. Using this feature decreases the attractiveness of denial-of-service attacks on the auctioneer, since a party of a previously defined size can restore the HMAC key and thus establish a new auctioneer in case the former auctioneer should not be available anymore. It should however be noted that using the secret sharing scheme would allow collaborating parties to restore the key and enable them to create money on their own. Whether this has to be considered an attack or not depends on the application and the trustworthiness of the bidders.

The credit, i.e. the amount of money $val$ available to peer $k$ at point in time $j$, will be denoted as $value_{k,j}$. The auctioneer generates new instances as follows:
value_{k,j} = (val, hmac[val, rand_j]_K_{hmac})

It should be noted that the use of an HMAC for providing authenticity and integrity of the distributed amounts does not prevent cooperating bidders from copying credit, since all credit is signed with the same key. If this would be considered an attack, it could be avoided by adding a nonce for each amount of money which is distributed to the bidders. We did however not integrate this feature into the current distribution mechanism, as the consequence would be a significantly higher overhead for re-establishing an auctioneer from a reconstructed HMAC key: it would not only be necessary to distribute the HMAC key amongst the bidders at start up, but also to provide the set of currently valid nonces after each individual negotiation round. Nevertheless, realising robustness against collaborating bidders would simply be a matter of replacing the reimbursement module.

5) Management of Policies and Preferences: Policies for negotiation are provided by applications installed on the bidders. As such, a service is needed to allow applications to manage their policies. Multiple applications may define identical policies. Policies are unique and grouped by type. During negotiation, peers negotiate policies of each type sequentially. Table I contains examples of policies for the video conference scenario given in subsection II-A.

In the scope of this paper, a policy describes a data structure binding attributes to values. Each policy of the same type consists of the same attributes. Each attribute attr_i is unique and has a value j ∈ N : 1 < j ≤ 10. For example, the policy “EncVideo” features the possibility to cooperate using encrypted video telephony. This provides the users to cooperate very efficiently. As such, the value for the attribute “Cooperation” is the maximum value of 10. Since the communication is encrypted, the parameter “Security” also has the value 10. However, encryption and video telephony require a lot of power, hence the value for “Energy Saving” is 1. Here, these values are just raw estimations for the sake of demonstration but in [18] we have shown how correct values for such attributes can be determined and used.

Users and/or applications on the device provide a preference pref_i for each attribute attr_i. Let P be the set of policies p_i available for every peer. This set is called the policy set. For each policy of the policy set, the peer's calculate a utility value utility(p_i) using a utility-function [19], [20]. This value is defined as:

\[
\text{utility}(p_i) = \frac{u(p_i)}{\sum_{i=0}^{n}(u(p_i))^{-1}}
\]

The value of the utility function is used to calculate the bids for each policy. Let value_{k,j} be the amount of credit peer k is in possession of at point in time j. An additional aggression parameter aggression_i ∈ N : 1 < i < 10 is used to regulate how much of his credit a peer may spend during one auction. The value is set by the user or aggregated from the applications which registered policies and may be used to express the interest a user or his applications have in a particular auction. Each bid b_{k,i} of peer k for policy p_i is calculated as follows:
It should be noted that the architecture does not enforce a certain encoding of policies. In resource-constrained scenarios a complex XML-based policy language might not be the best choice. To cope with this fact, different policy representations can be supported by implementing the Policy interface.

6) Wallet: As auctions are based on amounts of virtual money which is exchanged between the auctioneer and the bidders, each peer needs a way to locally store his available credit. The component “Wallet” provides a service for locally storing virtual money.

7) Reimbursement: For auctions, each bidder’s market power depends on the available credit. Besides calculating the result of the auction, the auctioneer is required to provide an initial amount of money to each bidder and to manipulate these values according to the auction model in use. While negotiation of policies and reimbursement of the peers are two processes which depend on the specific auction algorithm in use, decoupling them provides the developer with the ability to recombine different auction algorithms with different reimbursement schemes. The reimbursement function \( \text{reimburse} \) is defined as:

\[
\text{reimburse} : \left( \begin{array}{c}
\text{value}_{0,j} \\
\vdots \\
\text{value}_{k,j}
\end{array} \right) \rightarrow \left( \begin{array}{c}
\text{value}_{0,j+1} \\
\vdots \\
\text{value}_{k,j+1}
\end{array} \right)
\]

The implementation steadily increases the amount of money each peer will receive upon the next reimbursement procedure in the background. Furthermore, the framework decouples the actual reimbursement of the auction peers from the calculation of the new values. Two services are used to define this behavior, \text{ReimbursementService} and \text{RefundModifierService}. ReimbursementService it is only responsible for transferring the new currency values to the bidders and does not change even when implementing different auction algorithms. RefundModifierService on the other hand performs the actual computation of the new currency values which will be reimbursed to the client. Therefore it has to be implemented or at least configured for each type of auction algorithm individually. The selection of a specific \text{RefundModifierService} implementation and the configuration of additional parameters, like the reimbursement cycle duration, is done using XML files.

8) Negotiation: Every auction algorithm is designed and implemented in form of a separate component. The framework defines a standardized procedure to implement new protocols based on finite state machines. These are designed using a model driven approach. Only the negotiation itself is described using this component. For calculating prices resulting from different auction protocols and reimbursing peers the reimbursement component should be used. The relationship between negotiation and reimbursement is depicted by Figure 2.

The auction protocols depend on the previously described mechanisms for accessing remote services. Upon initialization, bidders wait for the availability of an auctioneer. Once an auctioneer is available, all bidders try to authenticate the auctioneer using credentials based on asymmetric cryptography which bind the auction service to a verifiable identity. These credentials are provided by the auctioneer.

Once the authentication process is completed successfully, the bidders start the auction state machine and provide an auction service to the auctioneer. This service provides access to registered policies and is furthermore used to get bids from the participants and to announce the result of an auction. All communication between the auctioneer and the bidders is confidential. In order to provide message authenticity, integrity and confidentiality, a symmetric session key is exchanged between each bidder and the auctioneer using the auctioneer’s credentials.

One-shot auctions, like the Vickrey auction, require each peer to provide exactly one bid. This is a desirable property since each message exchange takes time and drains the battery of mobile devices. Other protocols, like the English or the Dutch auction, usually require significantly more messages to establish the result since multiple bidding rounds are conducted. These protocols are less suitable for scenarios with resource-constrained devices and limited available bandwidth.

9) User Interface: Since the result of auctions is based on individual preferences, support for user interfaces is required to allow application developers using the framework to provide users with the ability to modify the behavior and to set individual preference values. In our prototype, we integrated frontends for bidders running on mobile devices and for the auctioneer, running on a normal desktop PC. While the
mobile device frontends provide comprehensive interaction with the framework, the auctioneer frontend features a rudimentary user interface to start and stop the host and to export an easily parseable history of the negotiation process. These frontends have just been created for demonstration purposes and application developers who use the framework might want to provide their own GUIs or even let the negotiation run without any possibility of user interaction at all.

V. PROTOTYPE

To show that our architecture is suitable for AmI scenarios we implemented it on a platform for resource-constrained mobile devices. Google Android provides an application framework based on a modified Linux environment. While Android does not restrict the choice of the programming language to Java, it is by far the best supported language. Therefore, the prototype has been implemented in Java.

A. Choice of a modularisation platform

After evaluating the Android application framework and its limitations, the decision was made to use the component- and service based OSGi Framework [15] for developing the prototype. This approach allows us to use the framework on resource-constrained devices, like Android-based mobile phones, as well as any other platform featuring a J2SE-compatible Java virtual machine.

The OSGi-Alliance describes an open standard for remote access to services in a distributed environment. Eclipse ECF and Apache CXF are implementations of these remote service specifications. During the development of the prototype, ECF was still in the early stages of adopting the specifications. Apache CXF is based on Web Services. As such, it requires a rather considerable software stack, which poses a problem on resource-constrained devices. The same argument holds for pure Web Services [21].

R-OSGi [22] is another implementation of remote services for OSGi which was designed to be used on resource-constrained devices. In addition, the developers provide an implementation of the service location protocol [23], which is used for service discovery. During the implementations, R-OSGi had to be modified to support the Dalvik VM’s class format.

B. Model-driven design of auction protocols

Auction protocols may be implemented using a model-driven approach based on Unimod [24], an Eclipse plugin for designing FSM-based application logic, such as stateful network or negotiation protocols. Several auction algorithms were implemented using this plugin. Unimod translates graphical state machine models into executable Java code. This approach simplifies the implementation and modification of auction protocols. Of course, other approaches may also be used to define negotiation protocols which use the framework.

Figure 3: Steps of an auction

Figure 4: Screenshots of the Prototype GUI, running on an Android Phone
C. Evaluation

The implementation of our prototype was tested on several Android-based mobile phones, such as the HTC Dream (Android 1.6) and the Motorola Milestone (Android 2.0). During testing we noticed that initialization of the OSGi framework and loading of the bundles needs a considerable amount of time (55.6 s on the HTC Dream, 36.8 s on the Motorola Milestone). This is mainly caused by Android’s bytecode verification and the dependency resolution in OSGi. We thus assume that this delay could significantly decrease by avoiding verification of each bundle at start up and by predetermining the wiring of OSGi services. Furthermore, we tested the all-pay auction protocol using three peers – two phones (HTC Dream and Motorola Milestone) and a Lenovo T400 Notebook. The operation needed for one auction round add up to 3.7 s when using AES in CBC mode with 128 bit block length and 1024 bit RSA key pairs. Our research confirms Capra et Al’s [12] results concerning auctions as a suitable approach for distributed policy negotiation. While the implementation comes up to expectations of a proof-of-concept application, the out-of-the-box use of Apache Felix on current Android phones does not provide enough performance to be considered for a real-life scenario.

VI. Conclusion and Future Work

In this paper, we proposed a framework architecture for the secure multi-party negotiation of policies in order to support self-protection in AmI scenarios. The architecture of our framework is device- and platform independent and focuses on the use of auction protocols for negotiating policies. The top priority was the security of the auction protocols. All secret information that is exchanged between the auction host and the peers is encrypted and protected against illegitimate modifications.

To show the architecture design is sensible, we implemented a prototype for Google Android and for the J2SE Desktop platform. The prototype features two parametrizable auction protocols which were designed in a model-driven way and added in form of modules to the proposed framework: an all-pay auction and a Vickrey auction protocol. Using our prototype implementation we were able to show that both protocols are suitable for a secure multi-party policy negotiation in the exemplary setting of a mobile collaboration platform. Altogether, from our experience we deem mechanism design in general and auctions in particular as a promising approach to cope with the heterogeneity of Ambient Intelligence applications.

As part of our future work, we will extend the modularity of the proposed framework in order to allow individual utility functions and a replaceable module for determining the winning policy. Further, the integration of our framework into the refinement process of semantic high-level policies is in progress.

References


