Adaptive security protocol selection for mobile computing

Bruno P.S. Rocha\textsuperscript{a,b}, Daniel N.O. Costa\textsuperscript{b}, Rande A. Moreira\textsuperscript{b}, Cristiano G. Rezende\textsuperscript{b,c}, Antonio A.F. Loureiro\textsuperscript{b,*}, Azzedine Boukerche\textsuperscript{c}

\textsuperscript{a} Eindhoven University of Technology, The Netherlands
\textsuperscript{b} Federal University of Minas Gerais, Brazil
\textsuperscript{c} PARADISE Research Laboratory, University of Ottawa, Canada

\begin{abstract}

The mobile computing paradigm has introduced new problems for application developers. Challenges include heterogeneity of hardware, software, and communication protocols, variability of resource limitations and varying wireless channel quality. In this scenario, security becomes a major concern for mobile users and applications. Security requirements for each application are different, as well as the hardware capabilities of each device. To make things worse, wireless medium conditions may change dramatically with time, incurring great impact on performance and QoS guarantees for the application. Currently, most of the security solutions for mobile devices use a static set of algorithms and protocols for services such as cryptography and hashes.

In this work we propose a security service, which works as a middleware, with the ability to dynamically change the security protocols used between two peers. These changes can occur based on variations on wireless medium parameters and system resource usage, available hardware resources, application-defined QoS metrics, and desired data "security levels". We compare our solution to some widespread static security protocols, demonstrate how our middleware is able to adapt itself over different conditions of medium and system, and how it can provide a performance gain in the execution of cryptographic primitives, through the use of data semantics.

\end{abstract}

1. Introduction

The tremendous advances in wireless data communication and mobile computing have created a new computing paradigm that promises to provide services anytime and anywhere for everyone. Such an environment enables users to access a wide range of services and applications using a large variety of mobile and ubiquitous devices. Voice and video streaming, file transfers, notification and localization are some examples of applications developed for this environment.

In this diverse, dynamic and complex scenario, it is not appropriate to create static specifications for ubiquitous applications. Designers and developers should consider, for instance, hardware limitations, data communication properties, QoS requirements, and security demands when designing services for pervasive environments. Unfortunately, all this information is not promptly available beforehand what leads to a very difficult and challenging design. In any wireless technology, risks are inherent. Some of them are similar to those of wired networks, some are more severe due to the wireless communication, whereas others are new. Probably the most important source of risks in wireless networks is the data communication medium, which is easily accessible to intruders. The loss of confidentiality and integrity, and the threat of denial of service (DoS) attacks are risks common to wireless communications. Unauthorized users may gain access to the system and information, corrupt data, consume network bandwidth, degrade network performance, launch attacks that prevent authorized users from accessing the network, or use legitimate resources to launch attacks on other networks.

Given all these potential problems, the specification of security protocols becomes a major concern. From the system point-of-view, each application may not only have different security needs, but also different QoS demands. Furthermore the application may be running on a hardware platform with different capabilities and communication protocols. Some of the current protocols for wireless networks, such as IEEE 802.11 and Bluetooth, propose solutions which are either incomplete or flawed (Karygiannis and Owens, 2002), besides been focused only on the link layer of the network stack. Even though updates have been made to improve native security of these protocols, in...
practice there is a great heterogeneity on which of these mechanisms are actually used, easily allowing misconfiguration. For instance, even the flawed WEP can still be found in operation (RSA Security, 2007).\footnote{For an example, as of October 2009, US-based Verizon FiOS internet provider installs home-based internet routers with WEP security by default.}

A common solution to this problem is the adoption of application-level security mechanisms originally developed for desktop computers and Internet applications. This approach is not always successful, as the challenges posed by mobile devices often create a “gap” between requirements and hardware capacities (Ravi et al., 2002). Besides, security mechanisms designed for typical Internet applications usually do not consider the distinct properties of the wireless medium, which are important information sources regarding the local environment (Li et al., 2006).

In order to overcome problems related to heterogeneity of hardware, software, and communication, it has become common the development of a mobile computing middleware. A middleware is a software layer that intermediates the interactions between applications and lower-level system directives, such as data communication. Its role is to provide the application with directives for transparent interaction with the underlying distributed system (Capra et al., 2001; Rocha et al., 2007).

In this work, we propose a context-aware security middleware that dynamically changes the security protocols used between a pair of peer-entities according to a set of variables. Our service, named AsecMid, monitors some parameters related to the wireless medium conditions, system resource usage, hardware capabilities, application-defined QoS metrics, and desired security services. Our solution transparently chooses the best security protocol for each transmission, from a large collection of protocols, according to the parameters monitored. The application accesses the middleware as a standard network socket, unaware of security protocols being applied. Besides, the application can supply a semantic description of the data to be transmitted, so that the sensitivity of each piece of data can be determined individually. The main purpose of this work is to bridge the gap between underlying network information, protocols, and primitives and the high-level service oriented software. The idea is not to develop new security algorithms, but use existing and consolidated protocols as efficiently as possible, based on the “Adequate Protection Principle” (Pfleeger and Pfleeger, 2006), which states that adequate security should be applied to each data type and context, independently. Our results show the effectiveness of our solution.

The main motivations for this work are as follows. First, there is an important aspect related to mobile computing security: the tradeoff between providing security and maintaining communication quality and system performance. Our service has the goal of working with this tradeoff, keeping the balance between security and performance according to the specifications defined by the supporting application. Second, the development of the service in the form of a middleware presents a benefit in providing a single security layer for different contexts of hardware, software and communication. Third, the characteristics of the medium where a device is inserted often are powerful sources of information (Li et al., 2006). With that, the utilization of parameters from different sources (e.g., application requirements, data semantics, wireless medium, system resources) is an interesting approach to create an adaptive and context-aware mechanism. Finally, the possibility of using data semantics to determine different sensibility levels of the data being transmitted can be interesting due to the fact that strong security mechanisms can be employed only where they are actually needed, providing an expected performance gain over utilizing the same mechanism for the whole data.

The main contributions of this work are:

- Formal definition of the problem of selecting security protocols based on security (cryptographic strength) and resource-usage (e.g., processing, memory and network overhead) metrics.
- Proposal of a methodology for treating the problem, divided in stages and also with the goal of not incurring considerable system overhead.
- Development of a reference implementation of the proposed mechanism, in the form of a middleware, ready to be executed on Linux environments and with portable source code.
- Evaluation of the solution concerning its adaptability regarding changes of the execution context (system resources and wireless medium conditions).
- Evaluation of the solution concerning its performance gain when a semantic description of the data being transferred is made available.

This paper is organized as follows. In Section 2, we present the related work. In Section 3, we discuss the environment model we considered, as well as definitions used throughout this paper, including a formal definition of the problem to be treated.

We discuss how to represent and deal with semantic data definitions in Section 4. The solution itself is presented in Section 5, including the system design and architecture, and the decision algorithms. Our experimental results are discussed in Section 6. Finally, in Section 7 we present our conclusions and future work.

2. Related work

Standard security mechanisms for wireless networks have been broadly studied, with its flaws discussed in literature. Patiyoot and Shepherd (1999) discuss the main cryptographic techniques for wireless networks, including IEEE 802.11, wireless ATM, GSM, UMTS and others. Karygiannis and Owens (2002) present a detailed study of the flaws and vulnerabilities of the native security services provided by IEEE 802.11 and Bluetooth. Bhagyavati et al. (2004) analyze security techniques for the IEEE 802.11 protocol and its variations (802.11i and 802.11x family). Denko et al. (2009) discuss, among other aspects, security issues in the design of a middleware for mobile ad hoc networks, and TalebiFard et al. (2010) discuss security and privacy in next generation IP-based networks, including wireless networks.

Some studies found in the literature have identified special approaches to promote security on wireless environments, pointing out challenges to achieve them. Some of these challenges include the problem of restricted processing capacity of mobile devices, as pointed out by Ravi et al. (2002) and Botha et al. (2009). Specific challenges regarding the IEEE 802.11 protocol are discussed by Arbaugh (2003) and Potter (2006), the latter focusing on hotspots.

Proposals have been made to provide security considering mobile and wireless properties. Soliman and Omari (2005) propose a cryptographic system aimed to the mobile domain, using flow cipher algorithms. Li et al. (2006) propose a system that uses radio signal properties for authentication and confidentiality on the physical layer of the network stack, outlining the necessity to work with cross-layer design in this scenario. Dong et al. (2009) propose two general frameworks that
encompass several network coding-based systems for wireless networks.

Tripathi (2002) and Denko et al. (2009) discuss requisites and challenges to design a middleware system. Capra et al. (2001) and Mascolo et al. (2002a) identify new requirements for a middleware that supports mobility, discuss the main differences between wired and wireless environments, and categorize and analyze some possible solutions for a mobile middleware. They also propose the use of a reflexive middleware for mobile computing (Capra et al., 2002), and describe an example for ad hoc networks (Mascolo et al., 2002b). Capra et al. (2003) present another middleware that allows context changes based on a set of rules.

Some examples of a security middleware are also presented in the literature. Aimed towards wired networks, Foley et al. (2004) propose a framework to interoperate between standard Web services middleware such as .NET, CORBA, and EJB. Focused on the wireless medium, Corradi et al. (2005) describe a middleware that deals with access control problems using mobile agents to access wired networks on behalf of mobile users which are out of reach or temporarily disconnected.

There are also some proposals in literature that consider the “Adequate Protection Principle” (Pfleeger and Pfleeger, 2006) and, thus, use different security levels according to each data part. Kim et al. (2005), for example, describe a protection schema for transmitting MPEG-4 videos to mobile devices in which MPEG video frames are transmitted according to their types, encrypting only most important frames. They study the effect of this selective cryptography on video exhibition and verify that a non authorized user cannot see its contents. This schema is similar to our data semantics description, but it is limited to MPEG video transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types. Zhou et al. (2010) propose and evaluate a security-critical scheduling scheme, which considers applications’ transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types.

Some examples of a security middleware are also presented in the literature. Aimed towards wired networks, Foley et al. (2004) propose a framework to interoperate between standard Web services middleware such as .NET, CORBA, and EJB. Focused on the wireless medium, Corradi et al. (2005) describe a middleware that deals with access control problems using mobile agents to access wired networks on behalf of mobile users which are out of reach or temporarily disconnected.

There are also some proposals in literature that consider the “Adequate Protection Principle” (Pfleeger and Pfleeger, 2006) and, thus, use different security levels according to each data part. Kim et al. (2005), for example, describe a protection schema for transmitting MPEG-4 videos to mobile devices in which MPEG video frames are transmitted according to their types, encrypting only most important frames. They study the effect of this selective cryptography on video exhibition and verify that a non authorized user cannot see its contents. This schema is similar to our data semantics description, but it is limited to MPEG video transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types. Zhou et al. (2010) propose and evaluate a security-critical scheduling scheme, which considers applications’ transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types.

Some examples of a security middleware are also presented in the literature. Aimed towards wired networks, Foley et al. (2004) propose a framework to interoperate between standard Web services middleware such as .NET, CORBA, and EJB. Focused on the wireless medium, Corradi et al. (2005) describe a middleware that deals with access control problems using mobile agents to access wired networks on behalf of mobile users which are out of reach or temporarily disconnected.

There are also some proposals in literature that consider the “Adequate Protection Principle” (Pfleeger and Pfleeger, 2006) and, thus, use different security levels according to each data part. Kim et al. (2005), for example, describe a protection schema for transmitting MPEG-4 videos to mobile devices in which MPEG video frames are transmitted according to their types, encrypting only most important frames. They study the effect of this selective cryptography on video exhibition and verify that a non authorized user cannot see its contents. This schema is similar to our data semantics description, but it is limited to MPEG video transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types. Zhou et al. (2010) propose and evaluate a security-critical scheduling scheme, which considers applications’ transmission, whereas our work proposes a generic mechanism for specifying security configurations for different data types.
In this way, the Diffie–Hellman scheme alone can be defined as a security algorithm. Other examples of algorithms include the DES for cryptography, SHA-1 for hash codes, any MAC code algorithm, or any key exchange scheme. Suppose two entities want to publish their RSA public keys using Diffie–Hellman to generate 128 bit symmetric keys and at each transmission they generate an MD5 hash code, encode it with the RSA private key, append it to the data packet and then encrypt everything with a triple DES using the leading 112bit of the symmetric key. This agreement just described between two entities provides an example of a security protocol.

3.3. Parameters and metrics

The proposed middleware works with different values in order to choose the most suitable security protocol. These values are divided into parameters and metrics and serve different purposes. Definitions 3 and 4 describe how these terms are used in this work.

**Definition 3.** A parameter is a value that holds information from the context in which the middleware is inserted, and which is used to aid the process of choosing security protocols to be used.

**Definition 4.** A metric is a value that defines a property of an associated security algorithm. Metrics can define both functional and security properties of algorithms. A security protocol metric represents the aggregation of that metric’s values for each algorithm of that protocol.

The middleware monitors parameters from different sources: wireless medium, hardware capacities, system resource usage, application-defined QoS constraints, and security levels. The parameters used in this work are listed in Table 1. The units aim to facilitate the protocol decision process, described in Section 5. Thus, percent units are most desirable, due to the fact that they have known upper and lower bounds.

Each security algorithm is associated with six properties defined by metrics. Of those, three are related to security strength (or cryptographic strength): confidentiality, integrity and authentication; and the other three are related to resource-usage: memory usage, tables for processing time, and network overhead. Security strength metrics are represented by integer numbers, between 0 and 100, whose utility is to compare different algorithms. As discussed in Section 5, these values are defined by the system administrator, throughout a configuration file. For example, AES cryptography should have a higher confidentiality value than DES, since it is conceptually stronger and safer. The difference between the same metric of two algorithms should represent how much stronger one is with respect to the other. It is out of the scope of this work to determine optimal values for the metrics. The correctness of our approach, presented in Section 6, is measured regarding the metrics only, and not which are the specific algorithms that hold them. Therefore, different values for the metrics might influence on which protocols are chosen, but not the validity of the approach. The resource-usage metrics are measured in bytes, for memory usage and net overhead, and microseconds for processing time. Network overhead and processing time are represented by tables, with entries for different data sizes. The process of obtaining these values is discussed in detail in Section 5.

It is important to notice that the focus of this work is to present and prove correctness of a mechanism that changes security protocols due to changes in the context parameters. Therefore, the choice of parameters and metrics can be altered without invalidating the results of this paper. The proposed parameters and metrics were chosen empirically, with the aim to represent the most important aspects of the wireless and mobile medium, mobile devices, execution context and algorithm characteristics. It is straightforward to adapt the proposed middleware to work with a different set of metrics and/or parameters.

From the presented lists of parameters and metrics it is easy to note that some parameters are related to certain metrics. We define this as a mapping that each metric has one or more context parameters. The formal definition of this mapping is presented below, which characterizes the problem. Table 2 presents which metrics are affected by each parameter considered in our work. For example, a high mobility value

<table>
<thead>
<tr>
<th>Source</th>
<th>Parameter</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless medium</td>
<td>Mobility</td>
<td>Percent</td>
</tr>
<tr>
<td></td>
<td>Channel quality</td>
<td>Percent</td>
</tr>
<tr>
<td></td>
<td>Latency</td>
<td>Microseconds</td>
</tr>
<tr>
<td></td>
<td>Routing</td>
<td>Direct or routed</td>
</tr>
<tr>
<td>Hardware capacities</td>
<td>Memory</td>
<td>Bytes</td>
</tr>
<tr>
<td></td>
<td>Radio information</td>
<td>Radio used (WIFI, Bluetooth, etc)</td>
</tr>
<tr>
<td>Resource usage</td>
<td>Memory consumption</td>
<td>Bytes</td>
</tr>
<tr>
<td></td>
<td>CPU usage</td>
<td>Percent</td>
</tr>
<tr>
<td></td>
<td>Battery (if present)</td>
<td>Percent</td>
</tr>
<tr>
<td>Application (QoS)</td>
<td>Maximum latency</td>
<td>Microseconds</td>
</tr>
<tr>
<td></td>
<td>Maximum memory usage</td>
<td>Bytes</td>
</tr>
<tr>
<td></td>
<td>Maximum net overhead</td>
<td>Bytes</td>
</tr>
<tr>
<td></td>
<td>Maximum negotiation overhead</td>
<td></td>
</tr>
<tr>
<td>Application (security levels)</td>
<td>Confidentiality level</td>
<td>Levels: disabled, optional, desired, mandatory or critical</td>
</tr>
<tr>
<td></td>
<td>Integrity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Authentication</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobility</td>
<td>Confidentiality and Authentication</td>
</tr>
<tr>
<td>Channel quality</td>
<td>Confidentiality, Integrity and Network Processing</td>
</tr>
<tr>
<td>Latency</td>
<td>Confidentiality, Authentication</td>
</tr>
<tr>
<td>Routing</td>
<td>Confidentiality, Authentication</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>Confidentiality, Integrity and Authentication</td>
</tr>
<tr>
<td>Radio information</td>
<td>Confidentiality, Integrity and Authentication</td>
</tr>
<tr>
<td>Memory consumption</td>
<td>Memory</td>
</tr>
<tr>
<td>CPU usage</td>
<td>Processing</td>
</tr>
<tr>
<td>Battery</td>
<td>Processing, Network and Memory</td>
</tr>
<tr>
<td>Maximum latency (QoS)</td>
<td>Processing</td>
</tr>
<tr>
<td>Maximum memory (QoS)</td>
<td>Memory</td>
</tr>
<tr>
<td>Maximum net overhead (QoS)</td>
<td>Network</td>
</tr>
<tr>
<td>Maximum negotiation overhead (QoS)</td>
<td>Authentication</td>
</tr>
</tbody>
</table>
implies in a higher number of possible devices in the vicinity, possibly increasing the necessities of confidentiality and authentication. However, mobility does not relate to memory usage, for example. As with the choice of parameters and metrics, the mapping between them is also fixed. The next Sections will describe how this mechanism works in more detail. Again, whether the mapping table is or not optimal does not interfere with the correctness of our approach, and it is out of the scope of this paper to pursue this kind of optimization.

We conclude the discussion on parameters and metrics with two examples of applications for the middleware. Our first example consists on a location tracking program that runs on specialized hardware on vehicles (cars, boats, etc.). The program makes a peer-to-peer communication (possibly via satellite) between the vehicle in question and a central server. In this scenario, the program treats confidentiality as a critical issue, since the transmitted data includes the vehicle’s location. Also, authentication is set to mandatory so that a such algorithm is always employed, avoiding an impersonation attack. Lastly, integrity is set to optional as it is not necessary, since the information is sent multiple times and errors are easy to detect. If the conditions favour it, though, its use is not discarded. As for the other parameters, wireless medium parameters tend to be extremely variable and hardware capacities tend to be somewhat limited. However, as this is a program running in dedicated hardware, the QoS parameters regarding resource-usage are not set to low values. That is, the application does not constrain much the middleware concerning resource-usage. Still, some loose limits for latency and overhead are imposed. In this scenario, the middleware will use the best confidentiality algorithm that its hardware allows and will always use some authentication algorithm. If in high mobility, it may tend to choose a protocol that dedicates some more processing time to a stronger authentication protocol. Also, if channel quality is low, it can choose to use a protocol that includes an integrity component.

The second example consists of a distributed audit program that runs as a background process on several computer elements (from desktops to PDAs) of a large company. The software’s purpose is to gather audit logs for several sensitive actions of its employees. It works on a peer-to-peer basis and, as the previous example, has some flexible requirements. For this program, integrity is mandatory as the correctness of the audit logs are its most important aspect. Confidentiality and authentication are set to desired as the information is not necessarily secret but company’s administrators think that, if possible, masquerading its communication is desired. The program can run on a multitude of wireless mediums and hardware capacities, and resource-usage can also vary a lot, as it is meant to be a background process. QoS parameters are then set to somewhat restrictive, ensuring the process will not degrade performance of its host machine. This program would then run with a security protocol in accordance to its host machine. An instance running on a router or a security camera would simply pick a very light security protocol, as resources are scarce. An instance running on a laptop would dynamically change its chose protocol according to the current machine’s resource usage and wireless medium conditions. On the machine’s idle times, stronger protocols would be used. In all cases though, an integrity algorithm is always used. These examples illustrate that the target applications for our middleware are the ones with somewhat flexible security requirements. Also, should the application have a pre-determined communication pattern, it can specify this pattern, outlining different security requirements for distinct parts of the information. This is explained in detail in Section 4.

3.4. Problem characterization

With the definitions presented above we characterize the problem we treat in this work. Firstly, we shall list all elements involved:

- A set of security protocols, according to Definition 2.
- Each protocol has a set of metrics, with six elements, discussed in Section 3.3.
- A set of parameters related to the execution context, discussed in Section 3.3.
- We know that each parameter is directly connected to at least one metric, according to Table 2.

With that, we can formalize the problem elements and their relationships in Definition 5.

Definition 5 (Problem elements). Let \( S \subseteq S \) be the set of available security protocols. Let \( T \subseteq T \) be the set of metrics. Let \( P \subseteq P \) be the set of parameters related to the execution context. Function \( V: P \rightarrow \mathbb{R} \) returns the current value of a given parameter. Let \( MP: T \rightarrow 2^n \) be a mapping between parameters and metrics, where \( MP(t) \) returns a set of parameters that affect metric \( t \in T \). We know that each metric \( t \in T \) is mapped to one or more parameters, such that:

\[
\forall t \in T : MP(t) \neq \emptyset.
\]

The existence or not of a mapping between a metric and a parameter is known a priori. Also, let \( MS: S \times T \rightarrow \mathbb{R} \) be a mapping between protocols and metrics to values.

We know that the set of metrics of each protocol is formed by the following metrics: confidentiality, integrity, authentication, memory, processing and network. For convenience, we will call the metrics \( c_i, i, a_i, m_i, p_i \) and \( n_i \), respectively. With that, we extend the concept of security protocol, applying it to the problem, described in Definition 6.

Definition 6 (Security protocol, on problem’s context). We know the metrics \( T \), i.e.,

\[
T = \{c_i, i, a_i, m_i, p_i, n_i\}
\]

where \( p_i \) and \( n_i \) are arrays, whose \( k \)-th element is accessed as \( p_i[k] \). With this, we know that for each protocol \( s \in S \), there are exactly six mappings with metrics. Lastly, we define that each of the six metrics has a normalized value, all in the same range, consisting of integer numbers. Metrics \( c_i, i \) and \( a_i \) represent the protocol’s cryptographic strength, with higher values being most desirable, while metrics \( m_i, p_i \) and \( n_i \) represent resource-usage, being lower values most desirable. If we specify \( Q(s) \) as the “quality” of a protocol \( s \), we can say that:

\[
Q(s) = \frac{MS(s, c_i) + MS(s, i) + MS(s, a_i)}{MS(s, m_i) + MS(s, p_i) + MS(s, n_i)}.
\]

However, it is known that cryptographically stronger protocols tend to use more system resources, creating the relation:

\[
\forall s \in S : MS(s, c_i) + MS(s, i) + MS(s, a_i) \sim MS(s, m_i) + MS(s, p_i) + MS(s, n_i).
\]

Thus, \( Q(s) \) tends to a same constant value for most of the cases.

From the previous definition we can then identify the problem to be solved. The goal of the proposed middleware is to select, in real-time, the “best” security protocol for a given context. However, it follows from the definition that protocols “quality” levels, with the proposed metrics, tend to have similar values for all the domain of protocols. With that, the information in Definition 5 about the mapping of metrics to context parameters becomes significantly important to solve the
problem. Therefore, we formalize the problem to be treated in this work in Definition 7.

**Definition 7 (Real-time security protocol selection problem).** Having a set $S$ of protocols, according to Definition 6, a set $P$ of context parameters, a set $T$ of protocol’s metrics and mapping relations $MP$ and $MS$ between metrics and parameters and metrics and protocols, respectively, according to Definition 5, choose the most suitable protocol for a determined context.

The approach taken, as well as generation of protocols and metrics, acquisition of parameters and other details are presented in Section 5.

### 4. Semantic data description

The goal of the semantic data description is to allow the application to configure desired security levels independently for each part transmission without the need to worry with details of how security is applied. That is, the application defines security and QoS requirements in a qualitative way, and the underlying middleware is responsible for considering security levels and applying the security method most applicable to the application needs. In this way, the application can define stronger security and QoS requirements for the most important transmissions, and even define that some parts of the transmission do not need any security guarantee at all.

In this work we define and use the LACS language (Language for Annotation and Configuration of data Security) to describe the semantics of data transmitted by applications (see Section 4.2). This language allows the definition of Security Configurations and the description of transmissions to be realized, as well as an association between transmissions and previously defined configurations. In the next, we define in more detail the Security Configurations and the LACS language.

#### 4.1. Security Configuration

First, we define the term Security Configuration:

**Definition 8.** A Security Configuration is a set of security and QoS requirements that define the semantic value of a block of data.

The security and QoS requirements, which are part of a Security Configuration, are the same application defined parameters presented in Section 3.3, excluding maximum negotiation overhead. They are:

- **Confidentiality level:** Determines the confidentiality of the data and, thus, it should not be read by a non-authorized entity that, by any means, intercept it. It can assume the values: disabled, optional, desired, mandatory and critical. The first level explicitly states that no encryption method should be used; the next two mean that it may or may not be used; and the last two mean that some form of encryption is strictly needed.

- **Integrity level:** Determines that the data should arrive at the destination exactly as it was sent, that is, with no modifications of any kind. It can assume the same values of the previous requirement.

- **Authentication level:** Determines that only authorized entities can access the data. It can assume the same values of the previous requirements.

- **Maximum latency:** Determines the maximum time, in microseconds, for the latency. This is from the application point-of-view, that is, time is related to the moments a packet leaves and arrives at the application layer itself.

### Maximum memory usage

Some security protocols incur in the utilization of additional memory space. This requirement defines a limit for memory utilization, in bytes.

### Maximum network overhead

Some security protocols include additional bytes on the data itself (such as hash codes or crypto algorithms that produce an output larger than the input). This requirement defines a limit, in bytes, for this additional overhead.

#### 4.2. Annotation language

The LACS (Language for Annotation and Configuration of data Security) is a language that allows the application to describe the desired security levels through Security Configurations, and define associations between these levels and each individual transmission. These descriptions are kept in an annotation file, in XML. This file describes the transmission model used by the application, that is, a description of the sequence of communications (sends and receives) to be done and the association of each transmission to a security configuration, indicating the desired security and QoS levels.

By supplying an annotation file, the application commits itself to transfer data in the same pattern described by the file. The non accomplishment of this pattern may result in transmitting data with inconsistent security levels. However, the application may at any time supply a new annotation file to the semantic configuration module (explained in the architecture section), redefining the transmission sequence. In that case, the sequence is restarted.

**Definition 9.** An Annotation File ($AF$) is defined as a tuple $AF=(D,T)$, where $D$ is the Declarations (descriptions of Security Configurations) and $T$ is the Transmission Model, which contains a pattern of the sequence of communications to be done (sends and receives).

The Declaration section ($D$) can, in turn, be divided into two parts: security and message definitions. The first contains the definitions of each Security Configuration used by the application, while the latter contains the definitions of some special messages that will be transmitted.

The security definitions are composed of a Global Configuration and one or more Security Configurations. The global configuration is the basis of the others. That is, all defined security configurations inherit all requirements defined in the global configuration. Similar to the inheritance concept of object oriented programming languages, the security configurations can redefine any requirement previously defined in the global configuration, prevailing in this case the definition from the security configuration. This mechanism is specially useful when there is a requirement appropriate to all transmissions.

Both the global configuration and the security configurations are made of a set of requirements (discussed in the previous section), which indicates the desired security levels, as well as QoS constraints. Listing 1 shows an example of security definition, in which the global configuration determines a restriction of maximum latency of 5 seconds to all transmissions. Besides, there are two security definition configurations: a low one with all security requirements disabled and a high one with them all at critical levels.

**Listing 1. LACS: Security definition example**

```
<security-definition>
  <global-configuration>
    <requirements>
      <maximum-latency priority="Medium" time="5000000"/>
    </requirements>
  </global-configuration>
</security-definition>
```
The set of message definitions allows the application to specify messages with special properties that it will either send or receive. These definitions are useful to compare if a received or transmitted message is of a certain type. LACS defines two kinds of message comparator which can be used together. The mask-comparator applies a bit mask over a defined set of bytes and compares it with a determined value. The size-comparator compares the message size with a determined value using a comparison operator (\(=,\neq,>,<,\leq\)).

Listing 2 shows an example of two message definitions. The first is a special message whose two first bytes have the value 0x0103. The second one is a message labeled "end-loop" whose characteristic is to be just 3 bytes length. Mask and size comparators can also be combined to define a message.

**Listing 2. LACS : Message definition example**

```xml
<message-definition>
    <message name="special-message">
        <mask-comparator first-byte="1" mask-size="2">
            match-mask="0xFFF" match-value="0x0103" /
        </mask-comparator>
    </message>
    <message name="end-loop">
        <size-comparator operator="equal" value="3" />
    </message>
</message-definition>
```

The Transmission Model (T) is a sequence of commands which describes the communication protocol used between two communicating entities, as well as the security configurations associated with each transmission. This description is made through commands similar to those of a procedural programming language. These commands can be sending or receiving messages, conditional commands ("ifs") that allow the execution of different command blocks according to some condition, or even loop commands that determine the repetition of a command block until a given condition is satisfied.

The following commands are supported by the LACS language:

- **Send**: Indicates the transmission of a message using the configuration determined by the attribute configuration. There are also the attributes label, to give a reference name to that transmission and size to make the semantic configuration module verify the message size before its transmission.

- **Receive**: Indicates the receiving of a message, with the exact same attributes of the send command.

- **Send-Receive**: Has the same attributes of the previous commands, but can indicate either a send or receive message, no matter which.

- **If**: Conditional command that allows the execution of different command flows depending on the result of the condition. It is followed by the commands: if-condition, if-then and if-else (optional).

- **If-Condition**: Set of conditional commands whose results determine the execution flow of the previous if condition. The result is true when all conditions are satisfied. Conditional commands can be: compare-message, receive-message or send-message. All explained in the following.

- **Compare-Message**: Compares a message (referenced through the attribute label) with a previously declared message on the Message Definitions section (indicated by the attribute name).

- **Send-Message**: Verifies whether it is receiving a pre-defined message, indicated by the attribute name.

- **If-Then and If-Else**: Followed by sequences of commands to be executed in case the previous if condition is true or false, respectively.

- **Loop**: Determines a command sequence that might be executed multiple times. There are two possible stop conditions (which can be used at the same time): a count attribute which defines a fixed number of iterations, or a stop-condition command, identical to the if-condition. The language also allows the definition of loops without stop conditions.

Listing 3 shows an example of a Transmission Model, in which the transmitted begins sending twice a high security message and receiving, for each of them, a 4 KB low security message. If the second sent message is a special message, the transmitter exchanges two more messages. Otherwise, entities exchange 10 pairs of low security messages, unless the transmitter receives an "end-loop" message, with which the low security sequence is interrupted. The whole process repeats itself.

**Listing 3. LACS : Transmission model example**

```xml
<translation-model repeat="true">
    <send configuration="high-security" />
    <receive configuration="low-security" size="4096" />
    <send configuration="high-security" label="second-msg" />
    <receive configuration="low-security" size="4096" />
    <if>
        <if-condition>
            <compare-message label="second-msg" name="special-message" />
        </if-condition>
        <if-then>
            <send configuration="high-security" />
            <receive configuration="low-security" />
        </if-then>
        <if-else>
            <loop count="10">
                <stop-condition>
                    <compare-message label="sent-msg" name="end-loop" />
                </stop-condition>
                <send configuration="low-security" label="sent-msg" />
            </loop>
        </if-else>
    </if>
</translation-model>
```
The Transmission Model repeats the command sequence by default. Should this behavior be not desired, it is possible to specify the repeat="false" attribute on the transmission-model tag, guaranteeing that the flow will not be repeated.

The Transmission Model, the communication is described between two entities, always from the point-of-view of one of them. The opposing entity interprets the commands in an inverse way, that is, sends as receives and vice-versa. The annotation file has to be identical for both sides.

5. Proposed solution

As previously discussed, the main goal of the proposed middleware is to dynamically choose the most adequate security protocol for communication between two peers, in a given context. The chosen protocol can be exchanged during runtime, should the context conditions vary. In general terms, the middleware seeks this goal by maintaining a library of existing security algorithms, evaluating them according to six defined metrics and monitoring a set of context parameters to be used in the decision making process.

In this section we discuss the proposed solution. Initially we present how the middleware can be configured by users, such as system administrators or applications. Afterwards, we present its architecture, and then the algorithms for the protocol selection logic. We also show how the middleware acts differently if it has or not a data semantic annotation file.

5.1. Middleware configuration

A configuration file is read by the middleware each time it is initialized. This file contains the library with possible security algorithms to be used. Each entry in the file represents an algorithm and information about it, including the values of the six metrics. The values for the resource-usage metrics are automatically calculated during middleware installation (discussed further). The behavior of the middleware can be configured in the following ways:

1. Editing the configuration file to exclude any algorithm that, by any reason, should not be considered for selection.
2. Editing the configuration file to change the values for the three cryptographic strength metrics of each algorithm, specifying how much an algorithm is stronger than another.
3. During runtime, the application can supply an annotation file describing security configurations for each part of the data, as explained in Section 4. If it is not possible to supply this semantic description, the application can define parameters related to desired security levels (confidentiality, integrity and authentication), as well as QoS constraints (latency, memory usage, network and negotiation overheads), for the whole execution of the application.

In this way, the application and/or system administrator can guarantee that the middleware will execute only trusted algorithms, and still define which of them is stronger. The application can also define minimum security levels independently for the three security services, as well as QoS constraints for each part of the data, via an annotation file that describes it semantically, or for the whole transmission, via API directives. For example, if at a given moment the confidentiality level is set to mandatory, that means the middleware will choose a protocol that includes data encrypting. Optionally, the application (both sides of it on the network) can also supply a password, which will be used to generate a cryptographic master key.

5.2. System design and architecture

The middleware is comprised of four fundamental parts: secure connection, security engine, parameter control and semantic configuration. The secure connection provides the API for the application to access the middleware. Its interface is very similar to a standard network socket library, and multiples instances of it may be run concurrently. The security engine is responsible for executing the decision making logic, as well as applying security protocols to packets sent or received by the secure connections. Parameter control monitors all parameters and makes them available for the security engine. Semantic configuration is only used if a secure connection is initialized with the corresponding semantic annotation file. In this case, the corresponding connection will have a semantic configuration module attached to it, responsible for processing the annotation file and providing security levels and QoS constraints relative to each data part. If the connection does not have an annotation file, global values for security levels and QoS constraints are supplied through the secure connection API. Fig. 1 illustrates the system architecture.

The parameter control module uses different methods to acquire each type of parameter. Wireless medium parameters are monitored by a separated software which accesses wireless card drivers. Medium information is kept on a buffer shared with the middleware. Hardware capacities and system resource-usage are obtained through operating system primitives, while QoS constraints and security levels work as explained above, been either supplied by the semantic configuration module (if annotation file is present) or by the secure connection API (otherwise). It is also important to observe that some parameters possess fixed values while others possess real-time values. Hardware capacities and global QoS constraints and security levels are fixed and need only to be acquired once. Semantic configuration QoS constraints and security levels are acquired for

Fig. 1. Architecture of the proposed middleware.
each individual packet. On the other hand, wireless medium information and system resource-usage are needed in real-time and are constantly monitored.

5.3. Security protocol selection logic

In order to make protocol selection easier, the middleware divides the process in stages. Computationally heavier tasks are done during the installation process. During runtime, some processing overhead is caused by the middleware initialization, as well as in the establishment of each new connection, whereas the decision making overhead for each transmission is kept to a minimum. Algorithm 1 presents a general view of the security engine execution flow (it does not consider the installation process). Each step is discussed individually in the following.

Algorithm 1. Basic security engine execution flow
1: Initializes security algorithms and protocols
2: Acquires fixed parameters
3: for all active connections do
4: Joins local and remote fixed parameters
5: Selects valid protocols
6: Selects a set of protocols to be used in the connection
7: Selects key exchange algorithms
8: Transmits protocol settings and key exchange algorithms to be used
9: Generates and exchanges necessary keys
10: while Connected do
11: if Send message then
12: Selects the best protocol, according to real-time parameters
13: Applies the protocol and concatenate its code and segments size information on the message
14: Sends the message to lower network layers
15: else if Receives message then
16: Reads protocol code and segments size information
17: Applies the protocol in “reverse” mode
18: Sends the message to the application
19: end if
20: end while
21: end for

5.3.1. Installation

During the installation process the middleware reads the configuration file, which in this point still does not possess resource-usage metrics for each algorithm, and then calculates the values of these metrics. For that, each algorithm is tested with different combinations of random generated data and keys, as well different sizes. The used sizes and number of executions for each of them can be configured beforehand. During these tests, the middleware measures memory usage, processing time and byte overhead over the execution and overhead added for each execution. For memory usage, the average of all executions is considered, while the tables for processing and overhead are filled with entries for each data size, considering the average for all executions of that size. Measurement results are then written to the configuration file, so that this process does not need to be repeated (unless the system undergoes a hardware modification, which would invalidate the previous results).

5.3.2. Initialization

When the middleware is initialized by the application its main task is to read and process the configuration file. With that, the middleware reads the algorithm definitions from the file and then generates a set with all possible security protocols (recall Definitions 1 and 2). The procedure for protocol generation respects the basic consistency rules, such as not generating a protocol which includes two algorithms from the same type (like a hash function). Besides, protocols which do not include algorithms from all three treated cryptographic services (confidentiality, integrity and authentication) are also generated, including protocols with only one algorithm. Key exchange algorithms are kept as separated algorithms and are not added to any protocol. They are treated in a slightly different manner, as discussed below. The system also generates a “special case” which represents the absence of security algorithms, as an “empty protocol”, for cases when plain transmission is needed.

Each protocol’s metrics are generated through the aggregation of the metrics for each individual algorithm. For each security metric it is considered the highest value between the current algorithms. For resource-usage metrics is made a sum of all costs of the individual algorithms.

Finally, during initialization, the parameter control acquires the fixed parameters. Hardware capacities are obtained from the operating system and in the phase the application either supplies a semantic annotation file (and thus instantiating a semantic configuration module for that connection) or global values for QoS and security levels.

5.3.3. Connection

For each established connection the middleware executes four basic steps: (1) combine fixed parameters, (2) select a set of protocols to be used during the connection, (3) select key exchange algorithms, and (4) generate and exchange keys. The first step is a simple process which consists in both peers transmitting their fixed parameters and keep, for each parameter, the most restrictive value.

After that, the middleware proceeds to select a set of protocols to be used during the connection. The goal of this step is to minimize the number of protocols to be considered during transmission, so that the real-time decision process is not very costly. In our experiments, discussed in Section 6, the initial set consists in more than 7000 security protocols, due to the large number of different possible combinations of algorithms.

The basic algorithm for selecting the protocol set, explained below, is executed a variable number of times. If the application supplies a semantic annotation file, the semantic configuration module supplies the set of all possible security configurations that will be used in that connection, and the protocol set selection will be executed for each security configuration, in order to choose protocols compatible with all combinations of QoS requisites and security levels. Otherwise, if the application supplies global values for QoS and security levels, the protocol set selection is executed only once.

The first part of the selection consists in eliminating protocols whose metrics do not satisfy fixed parameters constraints. This process only keeps protocols that satisfy all the restrictions below (remember the nomenclature of Definitions 5 and 6):

\[ S \prime = \{ s \in S \mid MS(s, m_1) \leq V(\text{mem.capac.}), \]
\[ MS(s, m_1) \leq V(\text{max.mem.}), \]
\[ \forall k \{ MS(s, p_k) \leq V(\text{max.latency}) \}, \]
\[ \forall k \{ MS(s, n_k) \leq V(\text{max.ovhd}) \}, \]
\[ (V(\text{conf lvl.}) \geq \text{mandatory} \Rightarrow MS(s, c_t) > 0) \]
\( (\text{int.lvl.}) \geq \text{mandatory} \Rightarrow MS(s, i) > 0), \) (6)

\( (\text{auth.lvl.}) \geq \text{mandatory} \Rightarrow MS(s, a_i) > 0), \) (7)

\( (\text{conf.lvl.}) = \text{disabled} \Rightarrow MS(s, c_i) = 0), \) (8)

\( (\text{int.lvl.}) = \text{disabled} \Rightarrow MS(s, h_i) = 0), \) (9)

\( (\text{auth.lvl.}) = \text{disabled} \Rightarrow MS(s, a_i) = 0), \) (10)

where, in case of a present semantic annotation file, the QoS constraints and security levels are checked according to one individual security configuration at each execution of this process. That is, equations from (3) to (10) are not checked at this point in the presence of a semantic annotation file. The restrictions, in the order the appear, discard protocols which: (1) consume more memory than the hardware’s capacity; (2) consume more memory than the maximum memory usage QoS restriction; (3) take more processing time than the maximum latency allowed, for all packet sizes; (4) incur a network overhead larger than the maximum allowed, for all packet sizes; (5), (6), (7) have no algorithm for confidentiality, integrity or authentication, while the corresponding QoS restriction states that the service is at least mandatory; and (8), (9), (10) have at least one algorithm for confidentiality, integrity or authentication, while the corresponding QoS restriction states that the service is disabled.

After this initial protocol elimination, the system adopts a greedy strategy to choose a small portion of protocols to be used during connection. At this point, the system chooses the following protocols:

- the \( N_1 \) with higher values for each security metric;
- the \( N_2 \) which spends less memory;
- the \( N_3 \) with less processing time and the \( N_4 \) with less overhead, for small data sizes (10 bytes);
- the \( N_5 \) with less processing time and the \( N_6 \) with less overhead, for large data sizes (128 KB);
- the \( N_7 \) with the larger sum of the three security metrics; and
- the \( N_8 \) with the smallest sum of the three security metrics.

To formalize this, we define \( S^N_i \) as the subset of security protocol set \( S \), containing the \( N \) elements with the highest value for metric \( t \). That is, \( S^N_i = \{ s \in S | s_i, s, i, 1, . . . , k : k \geq N, MS(s, t) > MS(s, t) \} \). Conversely, the opposite operator is defined the trivial way: \( S^N_i = \{ s \in S | s_i, s, i, 1, . . . , k : k \geq N, MS(s, t) < MS(s, t) \} \). If \( t \) is a sum of metrics instead of a single one, the operation is defined the trivial way, with the condition involving \( \sum MS(s, t) \) instead. With that, we can define the set \( S_i \) of security protocols to be used in the connection as:

\[
S_i = S^N_{c_i} \cup S^N_{b_i} \cup S^N_{a_i} \cup S^N_{m_{e_i}} \cup S^N_{m_{e_{i+10}}^j} \cup S^N_{m_{e_{i+10}}^j} \cup S^N_{m_{e_{i+128000}}^j} \cup S^N_{m_{e_{i+128000}}^j} \cup S^N_{m_{e_{i+128000}}^j} \cup S^N_{m_{e_{i+128000}}^j}
\]

(11)

The \( N \) constants are configurable and the used values are discussed in Section 6. In that way, a small group containing protocols with different metrics’ values is created. This choice is made only by one of the communication sides (the one that accepts the connection, by default), which then transmits the chosen set to the other side. Each protocol is numbered and a signature is generated for transmission.

The last decision to be taken during the connection phase is the choice of key exchange algorithms. This choice is done only once, regardless of the presence or not of a semantic annotation file. Firstly, the set of selected protocols in the previous step is analyzed in order to determine the number and types of keys needed. For asymmetric keys, a single mechanism is chosen, prioritizing the one that fits the requirement of maximum negotiation bytes and has the higher cryptographic strength (sum of security metrics). If the application provided a master key, it will be given priority the choice of an algorithm that uses it. For symmetric keys, a number of algorithms between one and the number of necessary keys is chosen, prioritizing algorithms according to the following order: (1) uses an existing master key and authenticates hosts, (2) uses an existing master key, (3) authenticates hosts, (4) uses asymmetric cryptography, and (5) the remaining ones. Algorithm 2 describes this process. The information of bytes used in the negotiation is a property typical of key exchange algorithms, being part of the algorithm information read from the configuration file.

Algorithm 2. Selection of key exchange algorithms

1: // Asymmetric key exchange
2: for all Asymmetric key exchange algorithms do
3: if Algorithm has negotiation overhead inside QoS constraint (global only constraint) then
4: if (There is a master key and algorithm uses it) or (there is no master key and algorithm does not need it) then
5: if Algorithm has higher cryptographic strength than current one then
6: Select algorithm as current asymmetric key exchange algorithm
7: end if
8: end if
9: end if
10: end for
11: // Symmetric key exchange
12: while Number of chosen algorithms ≤ number of needed keys do
13: // condition above is checked at each algorithm selection
14: if There is a master key then
15: Selects algorithms that use the master key and authenticate hosts
16: Selects algorithms that use the master key but do not authenticate hosts
17: end if
18: Selects algorithms that authenticate hosts
19: Selects algorithms that use asymmetric encryption
20: Selects the remaining algorithms
21: end while

Finally, the side that made the decisions sends the set of protocols and key exchange algorithms to the other side and then they distribute the keys. Key exchange algorithms are executed one by one, according to the order they were selected, to generate each of the necessary keys sizes. If the number of algorithms is smaller than the one of necessary keys, the list of algorithms is iterated in a cyclic fashion, restarting from the beginning. If the application has supplied a master key, the middleware uses it to encrypt all negotiation traffic with a pre-defined algorithm (AES, by default). This is done to avoid intruders from masquerading being a middleware instance, although this kind of protection is outside the scope of this work. Initialization vectors (IVs) are generated like keys, i.e., the algorithms chosen for symmetric key exchange will also be used to generate them. For the sake of simplicity one key and one IV is generated for each necessary size, according to the algorithms in the selected protocol set. That is, two algorithms that use a same type and same sized key (e.g. triple DES and AES with 128 bit key) will share both the key and the IV. This schema can be made more sophisticated in a relatively easy manner. Since key management
falls outside of the scope of this paper, the simpler approach was taken.

5.3.4 Transmission

The last step is executed for each data transmission. When a peer decides to transmit a packet of size \( k \), the middleware analyzes the real-time parameters to choose the best protocol for that moment, applies it over the message and concatenates the generated packet with the number of the used protocol and sizes of each segment, so that the other side can reversely apply the same protocol. At this point, the middleware possesses a small set of protocols (number depends on \( N_i \) constants and number of security configurations defined in semantic annotation file, if present), each one with its six metrics, and the set of parameters that can influence which metrics should be more important for that situation. The fact of this decision be critical in terms of time characterizes this part of the process as an online algorithm (Borodin and El-Yaniv, 1998), making unfeasible the use of computationally heavy techniques, such as ones based on optimization algorithms. For a quick and effective computation, our method obtains a score for each protocol. Given the linearized values of each metric, the score \( X \) of a protocol \( s \) is calculated as:

\[
X(s) = (M(s,c_i) \times W(c_i) + M(s,a_i) \times W(a_i)) - (L(s,p_i[k]) \times W(p_i) + L(s,n_i[k]) \times W(n_i) + L(s,m_i) \times W(m_i))
\]

where \( W(t) \) is the weight of metric \( t \in T \), as explained in Definition 6, and \( L \) is a linearization function that guarantees that all resource-usage metric values represent a percentage. This function is defined below.

\[
L(s,p_i[k]) = \frac{M(s,p_i[k])}{V(\max(latency)) - V(latency)}
\]

\[
L(s,n_i[k]) = \frac{M(s,n_i[k])}{k}
\]

\[
L(s,m_i) = \frac{M(s,m_i)}{V(\max(mem.))}
\]

Before calculating the weights, however, the middleware must check if each protocol is applicable to real-time conditions. For example, the overhead caused by a protocol may be under QoS restrictions for data sizes up to 10 KB, but over them otherwise. In case of no semantic annotation file, the middleware just selects protocols that satisfy the conditions below. Otherwise, the middleware acquires, from the semantic configuration module, the exact security configuration to be applied to the current message. It then acquires QoS constraints and security levels from the configuration, re-checks conditions (3) through (10) (protocols selected for different configurations are also present in the set), and filters protocols that satisfy the following conditions:

\[
S_c = \{ s \in S | M(s,m_i) \leq \min(V(\max(mem.)),V(\free(mem.))) \}
\]

\[
M(s,p_i[k]) + V(\text{latency}) \leq V(\max(\text{latency}))
\]

\[
M(s,n_i[k]) \leq V(\max(\text{ovhd}))
\]

If there is no \( k \) entry in the tables for processing and overhead, a simple interpolation with the values of the two closest entries (upper and lower) is made. These conditions are specific for the moment of transmission, since they consider real-time parameters. They exclude protocols which: consume more memory than either the system total or current free memory; 
take more processing time for the current packet’s size than the remaining latency allowed, i.e. the maximum supported latency minus the current network latency; and incur a network overhead for the current packet’s size larger than the maximum allowed.

After that each one of the six metrics has its value linearized (defined in an interval between 0 and 100). Security metrics already possess values between 0 and 100 by definition. Processing time is defined as the percentage of “remaining time” available for the protocol, i.e., the ratio between the entry on the processing time table and the difference between the maximum latency and current network latency. Network overhead is linearized as the percentage of the current message’s size, and memory usage as percentage of current available system memory or maximum memory QoS constraint (whichever is the lowest). This is defined by the previously mentioned I function.

The next step is determining each one of the weights of the metrics, used on the calculus of protocols’ scores. Knowing the mapping between metrics and parameters (Table 2), the value of each parameter is multiplied by a constant, which represents the weight \( K(p,t) \) of parameter \( p \) over the weight \( W(t) \) of metric \( t \) over the final score \( X \). In that way, the weight of a metric \( t \) is calculated by the following equation, where \( K(p,t) \) is the constant that represents the impact of parameter \( p \) over the weight of metric \( t \):

\[
W(t) = \sum_{p \in T} V(p) \times K(p,t)
\]

With the weights properly calculated, the middleware iterates over the set of protocols with the goal of finding the highest score. Although the whole process may seem complicated, it consists in only calculating values through simple arithmetics and then iterating over a small set of protocols, discarding non-suitable ones and keeping track of the one that has the highest score. Constants \( K \), like the \( N_i \) ones, are configurable and the values utilized are discussed in the next section.

5.4 About the security provided by the middleware

It is known that the true security provided by a security mechanism is defined by its weakest component. In the case of ASecMid, its security would then be defined by its weakest protocol used. This is in fact true, but it is important to point two things: (1) the set of protocols to be used is configurable, and this set should contain only protocols suitable for the specific application of the middleware; and (2) it is not the scope of this work to present the strongest security middleware. If this was the case, the solution would not be adaptive, but static, using only the best security algorithms for the specific application. Our work aims to provide an adaptive middleware, which will work on the tradeoff between security and performance, bounded by a configuration which is tuned for the specific application.

Another question is the possibility of an attacker to degrade wireless medium conditions to a point which the middleware uses only the weakest protocols. This is an attack that our work does not treat, but once again some remarks should be done: (1) as stated in the last paragraph, the configuration should ensure that the weakest protocol is still suitable for the specific application; (2) this kind of interference attack is also possible on static security mechanisms i.e., the possibility of suffering such attack has not arisen from our middleware design; and (3) even if this attack happens, depending on the attack intensity and on the configuration, ASecMid tends to resist the attack longer than a static mechanism, since the middleware would adapt itself, choosing protocols that can work even on the interference conditions, until a point in which no protocol can be chosen, as opposed to a static solution.

As already mentioned, it is out of the scope of this work to treat security of individual algorithms. This also includes cases in which configuration allows for a protocol where all algorithms are
individuals safe, but their specific combination causes some
security threat to arise. It is up to the middleware configuration
to prevent these cases from happening, although a simple
mechanism for avoiding use of specific algorithm combinations
can be easily implemented.

6. Experimental results

The great number of variables involved in this work makes
experimentation a difficult task. Besides, the use of a simulator
such as NS-2 also presents some problems, since it would be
necessary to acquire much information from both an operating
system and a wireless medium, and it is hard to find a simulator
that provides all necessary parameters (NS-2, for instance, does
not simulate the entire system’s resource-usage).

Therefore, we have adopted a different strategy: make real
executions of the middleware, but using simulated parameters. As
explained in the previous section, the simulation mode of
operation of our implementation is capable of creating different
profiles for parameters simulation. As the parameters are
simulated, they do not cause any real harm in the communication
being made. Thus, for each experiment we analyze the relations-
ships between parameter variation, distribution of used protocols
(as well as their cryptographic strength) and communication
throughput. Throughput analysis is directly related to how safe
the protocols being used (safer protocols tend to possess more
algorithms, and being computationally heavier) and, thus, have
two important meanings: (1) the middleware should have a
variable throughput over time, reflecting dynamic changes of
protocols, and (2) middleware’s throughput should be propor-
tional to how strict context conditions are, that is, in extremely
strict simulated scenarios it should employ “lighter” protocols,
having larger throughput, while in simulations of many available
resources it should employ “stronger” and thus slower protocols.
The parameters are varied in identical way in both communica-
tion sides.

In the next sections we present our implementation, an
analysis over the protocol selection overhead, and the results of
our experiments.

6.1. Implementation

We have made a reference implementation of the proposed
middleware, with the purpose of making simulations and serve
as a baseline for possible implementations in different devices. The
implementation was done in the C+ + object-oriented program-
ing language. Each module was implemented as a class, and the
parameter control is defined as a singleton (Gamma et al., 1995).
System data acquisition is done through directives of the Linux
operating system.

Regarding the K and N constants, we have a total of 24 values.
They interfere in the middleware’s decision making process, but
not independently, meaning that the value of one constant may
impact the value of the other. Besides, they are influenced by the
way parameters vary. With that, it is unfeasible to make experiments
trying to find “optimal” values for the constants.

What was done, in this case, we did manual executions and an
empirical treatment in order to find good values for each constant.
The values used in the implementation can be changed through
compilation directives. Table 3 shows the values used for the N
constants, presented in the connection part of Section 5.3. Table 4
shows the K constants, of the transmission part. Notice that some
parameters are not mapped to any metrics, that is because they
are used only in the restrictions (Eqs. (3) through (10), and (16)
through (18)), and some parameters have discrete values
(e.g., security levels) because they have a K value for each possible P.

The implementation has two modes of operation: execution
and simulation. The first one is the standard. In the latter
parameters are read from a simulation module, which in turn
reads an input file to determine how each parameter will be
simulated. It also has a separate module, which calls the
middleware using a traffic generator with some different genera-
tion patterns. Another difference happens when the middleware
is unable to select a protocol which obeys current QoS constraints
and security levels. When this happens in execution mode, the
middleware returns control to the application, reporting a
“requirements deadlock” exception. In simulation mode,
however, the middleware simply transmits the message in plain
text, without security protocols, and logs the moment when the
deadlock happened. Simulations are described in more detail
below.

Low level cryptographic primitives were used from third party
libraries.2 Our implementation has a library which consists of 90
security algorithms and, during initialization, generates 7755
protocols. It is good to remember that is out of the scope of this
work to discuss strengths and weaknesses of individual
algorithms. As mentioned in Section 5, the list of algorithms to
be used by the middleware may be previously configured. Our list
includes algorithms such as:

- Cryptography: Variations of DES, 3DES, AES, Camellia, RC4, RC2,
  CAST5, RSA, ECC, …
- Integrity: MD5, MD4, SHA-1 (and its variations), flavours of
  HMAC and CMAC, …
- Authentication: DSA/DSS, variations de private key encryption
  with RSA and ECC, …

com/products.html#borzoi.}

<table>
<thead>
<tr>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants N for protocol set selection.</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
</tr>
<tr>
<td>$N_1$</td>
</tr>
<tr>
<td>$N_2$</td>
</tr>
<tr>
<td>$N_3$</td>
</tr>
<tr>
<td>$N_4$</td>
</tr>
<tr>
<td>$N_5$</td>
</tr>
<tr>
<td>$N_6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants K for real-time protocol selection.</td>
</tr>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Mobility</td>
</tr>
<tr>
<td>Channel quality</td>
</tr>
<tr>
<td>Latency</td>
</tr>
<tr>
<td>Routing</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Radio</td>
</tr>
<tr>
<td>Memory usage</td>
</tr>
<tr>
<td>CPU usage</td>
</tr>
<tr>
<td>Battery</td>
</tr>
<tr>
<td>QoS constraints</td>
</tr>
<tr>
<td>Security levels</td>
</tr>
</tbody>
</table>

6.2. Protocol selection overhead

Since the protocol selection is done in stages, the decision overhead is kept to a minimum. The overhead of the installation phase is not important since it is taken only once per device. It represents multiple test executions of all algorithms considered. The number of executions depend on configuration. Being \( n \) the number of algorithms registered, the overhead of the initialization phase has a worst case time complexity of \( O(n^3) \) for generating every possible protocol (since each protocol has at most 3 algorithms) and keeping them in memory. This overhead also has little importance, since it is only taken once during the application loading. In practice, however, even though if \( n^3 \) comparisons are made, only a fraction of those generate valid protocols. In our experiments, for \( n \) equals to 90, 7755 valid protocols were generated, a number far below \( n^3 \) (more than 700 K). Since a protocol in memory is just a pointer to one or more algorithm descriptions, the space complexity for the protocols in memory is \( n_{kp} + n_{rp} \) where \( s \) is the average size of an algorithm specification, \( n_p \) the number of protocols generated and \( p \) is the size of a pointer in memory. Since the values of \( s \) and \( p \) are respectively 100 (approximately) and 4 bytes, we have a memory usage of around 40 KB for a library with the previously considered numbers. Acquisition of fixed parameters has constant complexity. During the connection phase the set of protocols to be used is chosen using a greedy technique, previously described, which is implemented with complexity \( O(n_{kp}) \), since it makes a fixed number of comparisons for each protocol. For each transmission, the overhead of the decision process is \( O(m) \), where \( m \) is the number of protocols chosen on the connection phase, which is basically equal to the sum of the values of the \( N \) constants, i.e. \( m = \sum N \), which tends to be a small number, close to a dozen in our tests. Finally, it is important to notice that the decision complexities for connection and transmission are also negligible due to the fact that they are taken together with network accesses. Thus, the time taken to compare e.g., 7000 protocols during a connection is minimal compared to the network access time (a loop with 7 K iterations versus the I/O waiting time for the network latency involved in a connection). The same is applied to transmissions, with the addition that the decision loop will iterate over less objects (typically a dozen).

6.3. Results

In the next sections, we present the experimental results. First we evaluate ASecMid’s (our middleware) capacity of adaptation. For that, we do not supply it with a semantic annotation file, since this feature is intended for performance gain only. In those experiments, we run ASecMid in different scenarios compared to executions without any security mechanisms and with emulations of four static widespread mechanisms: PGP and S/MIME for e-mail security applied to transmissions, IPSec, and the SET protocol for banking transactions. Then, we compare executions of the middleware with and without a semantic annotation file, in order to evaluate how much performance is gained by the use of the selective cryptography solution with the semantic analysis. Each experiment was executed 33 times, and the points represent their averages. For the results that show, error bars represent 90% confidence intervals, using the Z statistical test.

6.3.1. Capacity of adaptation

In this section each experiment is presented in four graphics, all with execution time, in seconds, in the \( x \) axis: the first shows the throughput variation of the considered mechanisms in the \( y \) axis; the second is a zoom of the first one, without the curve relative to the plain text transmission; the third shows the variation of some simulated parameters on the \( y_1 \) axis (left side), and the cryptographic strength of the middleware’s employed protocols on the \( y_2 \) axis (right side); and the fourth compares the middleware throughput on \( y_1 \) axis with its cryptographic strength on \( y_2 \) axis.

A typical execution happens with wireless medium parameters and resource-usage suffering a pseudo-random variation, that is, each parameter assumes a random value for a random time period, and then the next value and time period are recalculated, within acceptable limits. The only exception is the battery level, which has a decreasing value along the time. Fig. 2 shows this kind of experiment, with data traffic also being pseudo-random (each side sends a burst with a random number of packets, between 1 and 15, with random sizes between 1 and 64 KB). The variation of the parameters, with the decreasing battery, can be seen in Fig. 2(c). It can be seen in Figs. 2(a) and (b) how the middleware possesses a highly variable throughput, differently from static mechanisms. This is due to the fact that, throughout the execution, eight different security protocols were employed. In this experiment the application configured the middleware with strict global QoS constraints and security levels, in such a way that the protocols used by the middleware were “lighter” than the ones of the static mechanisms, causing middleware’s throughput to be larger. It is also important to notice in Fig. 2(d) how cryptographic strength (sum of the three security metrics of the protocol being used, varies from 0 to 300) has an inverse relation with throughput. As expected, this means that stronger protocols are computationally heavier, implying in less throughput, and vice-versa.

We present now an example of execution with a more controlled parameter variation. In the execution presented in Fig. 3, the global security levels and QoS restrictions were kept in low levels, several parameters had their values fixed and only system resource-usage (CPU and memory usage) increased over time. Fig. 3(c) shows how the middleware selected lighter protocols (decreasing the cryptographic strength) as long as resource-usage was increasing (memory usage, omitted in the graphic, varies in identical manner to CPU usage). Figs. 3(a) and (b) also show how throughput raises with time, reflecting the usage of lighter protocols. Fig. 3(d) show the variations of throughput and cryptographic strength. It can be seen that the throughput varies in a more continuous way, whether cryptographic strength varies in discrete steps. This happens because the strength is a human-defined metric, with some valued being more closely defined, while throughput is an observed measurement. Four different security protocols were used during execution.

We now show an example with other type of parameters varying over time. In the experiment of Fig. 4 the conditions are the same from the previous one, except that now the resource-usage parameters are fixed (with low usage), while wireless medium conditions (channel quality, mobility and latency) are deteriorating with time. From the mapping of parameters and metrics, we know the wireless medium parameters are related to both security and resource-usage metrics (Table 2). With that, a drop in medium quality may
Fig. 2. Typical execution, with high QoS restrictions and security levels, and pseudo-random variation of parameters. (a) Throughput compared to static mechanisms. (b) Zoom of throughput compared to static mechanisms. (c) Variation of middleware's parameters and cryptographic strength. (d) Variation of middleware's throughput and cryptographic strength.

Fig. 3. Execution with low QoS and security restrictions, and resources depleting over time. (a) Throughput compared to static mechanisms. (b) Zoom of throughput compared to static mechanisms. (c) Variation of middleware's parameters and cryptographic strength. (d) Variation of middleware's throughput and cryptographic strength.
influence the employed cryptographic strength with either a raise (security conditions deteriorate) or a drop (network conditions deteriorate). In this case, middleware's behavior is determined mainly by QoS restrictions. As in this experiment they are almost null, the middleware tends to employ stronger protocols. This can be observed in Fig. 4(c), which shows cryptographic strength raising as medium conditions deteriorate. In Figs. 4(a) and (b) we can notice how throughput drops a lot with the use of really strong protocols. Throughout execution time, three protocols were used, including one with the cryptographic strength of 281, the strongest generated with our library, which encrypts packets with the RSA algorithm using a 2048 bit key, calculates a CMAC code using AES with 256 bit key, and also a DSA authentication code. Fig. 4(d) shows the raise on crypto strength opposed to the drop on throughput.

If we raise QoS constraints and security levels from the previous execution we can observe a different behaviour. In this case, depicted in Fig. 5, the middleware tends to give a higher priority to resource costs as the medium conditions get worse. With that, we can see in Fig. 5(c) how the used protocols are getting weaker. From about 35 seconds of execution, however, the middleware cannot find anymore any protocols suitable for the high QoS restrictions and security levels required, and so the "requirements deadlock" happens. With that, the middleware starts transmitting in plain text, and throughput becomes the same of the plain transmission, as shown in Figs. 5(a) and (b). Before the deadlock event, four different protocols were used. In this experiment it was used an FTP traffic simulation, the sender side transmits data packets of 64 KB, and used protocols different from the ones used by the receiver (sends 1 byte ack messages). This happens due to the fact that protocols cause different processing times and network overheads for different data sizes.

Fig. 5(d) shows the opposed variations of crypto strength and throughput.

6.3.2. Performance gain with semantic data analysis

In this section we present experiments with and without a semantic annotation file supplied by the application, in order to compare the performance gain of this feature. In each experiment we compare four executions of the middleware: one with the annotation file and three others without it, but with different global configurations for QoS and security levels restrictions: one with strong security restrictions, one with strong QoS restrictions, and one with medium on both. Each experiment is presented in two graphs: the first shows throughput variation, while the second one shows cryptographic strength variation, both over time.

In the experiments of Fig. 6, it was used an FTP transmission. The annotation file describes the transmission classifying 64 KB packets as of high security importance, while 1 byteacks are considered with no importance. In the experiments of Figs. 6(a) and (b) all parameters are fixed, while the ones of Figs. 6(c) and (d) vary in a pseudo-random way with decreasing battery. As the semantic analysis execution uses plain text transmission for ack packets, it applies, in average, a smaller cryptographic strength than the others, obtaining a higher performance, but still applying the desired security level to important packets.

In the experiments of Fig. 7, it was used a traffic pattern which alternates between important and non-important packets, from the security point-of-view. Figs. 7(a) and (b) show the experiment with fixed parameters, while Figs. 7(c) and (d) show the experiment with pseudo-random parameters and decreasing
Fig. 5. Execution with high QoS and security restrictions, and medium quality deteriorating over time. (a) Throughput compared to static mechanisms. (b) Zoom of throughput compared to static mechanisms. (c) Variation of middleware’s parameters and cryptographic strength. (d) Variation of middleware’s throughput and cryptographic strength.

Fig. 6. Execution of FTP data transmission. (a) Throughput over time (fixed parameters). (b) Cryptographic strength over time (fixed parameters). (c) Throughput over time (pseudo-random parameters). (d) Cryptographic strength over time (pseudo-random parameters).
Fig. 7. Execution of a data transmission that cycles between messages with and without security importance. (a) Throughput over time (fixed parameters). (b) Cryptographic strength over time (fixed parameters). (c) Throughput over time (pseudo-random parameters). (d) Cryptographic strength over time (pseudo-random parameters).

Fig. 8. Execution of MPEG data transmission. (a) Throughput over time (fixed parameters). (b) Cryptographic strength over time (fixed parameters). (c) Throughput over time (pseudo-random parameters). (d) Cryptographic strength over time (pseudo-random parameters).
battery. It is easy to observe how the semantic analysis alternates between heavy protocols and direct transmission.

Finally, in the experiments of Fig. 8, we reproduce the traffic pattern of the MPEG video format (Le Gall, 1991). As in the previous experiments, Figs. 8(a) and (b) show the experiment with fixed parameters, and Figs. 8(c) and (d) show the experiment with pseudo-random parameters and decreasing battery. In these experiments the semantic annotation identify MPEG frames of types I, P and B, of high, medium and low security levels for each of them, respectively. Again notice how semantic analysis provides higher performance without compromising the offered security, in cases in which it is possible to determine previously the annotation file with semantics of data to be transmitted.

7. Conclusions and future work

In this work we presented a technique for an adaptive security protocol selection, in the form of a middleware, based on context conditions related to wireless medium, system resource-usage, hardware capacities, application defined restrictions, and data semantics. We also presented a reference implementation used in our experiments, which allows to configure the weights of decisions. The proposed middleware is also configurable by the application that uses it (QoS and security restrictions), as well as by the entity that maintains the system, throughout configurations in the algorithms library. As far as we know, this is a new approach to the problem.

Our results show that the proposed technique is adaptive to several contexts and leads to benefits both in performance and ease to use, since the application does not have to concern with applying security algorithms and protocols. The main factors that stimulate the use of the middleware are the presence of high variability of context parameters, and applications that allow some flexibility in the security protocol selection.

Besides, we have tested our middleware using semantic annotation files, which allow the application to determine different semantic values for each part of the data to be transmitted. With that, the adaption power of the middleware adds itself to the possibility of applying security according to the real sensibility of each data part. In this way, the middleware achieves a considerable gain in performance when it is possible to describe the data format to be transmitted.

The proposed middleware presents a new approach to an increasing security problem in the field of mobile computing. Besides, the proposed solution is applicable to a wide plethora of hardware devices.

One of the immediate future work is a more detailed treatment of the parameters. New parameters may be considered, such as the effective transmission rate, with the middleware changing protocols with the purpose of maintaining it in a determined level. The determination of constants $N$ and $K$ is also a point of interest, and techniques may be studied on how to better choose values appropriate for specific contexts.

From the architectural point-of-view, we have identified some points to be explored. One of them is the transference of the middleware to lower network layers on the protocol stack, and so deriving a technique for low level protocol switching, possibly on hardware routers. Another is the use of reflection techniques so that the middleware identifies and applies new security algorithms without changes in the source code. Finally, the protocol selection process can be done in a bidirectional fashion, possibly adopting a separated communication channel dedicated to the real-time negotiation.

This work also paves the way to be used as a basis for new security approaches. For example, the basic mechanism of protocol switching may be modified to support an environment of multiple hosts in which routing occurs. In this way, the middleware chooses and applies protocols for each peer-to-peer connection. An intelligent routing scheme for safe packets may be done, with packet re-encryption occurring in network areas in which the context is different. Fig. 9 illustrates this possibility.

The work can yet be applied to group security. A scheme for communication with distant network elements can be done, maintaining protocol negotiation only amongst neighbors, or keeping a single negotiation for a trusted group. Questions regarding mobile groups, acceptance of new members, trust and interactions between different groups are all also interesting.

References


