DR13.0: PROMISE End-to-End Security White Paper

Written by:

Kary Främling (HUT), David Potter (INDYON GmbH), Katja Seidler (SAP), Mario Neugebauer (SAP), Hong-Hai Do (SAP)

DELIVERABLE NO	DR13.0: PROMISE End-to-End Security White Paper
DISSEMINATION LEVEL	**PUBLIC**
DATE	12 December 2007
WORK PACKAGE NO	WP R13: PROMISE End-to-End Security
VERSION NO.	1.0
ELECTRONIC FILE CODE	document2
CONTRACT NO	507100 PROMISE A Project of the 6th Framework Programme Information Society Technologies (IST)
ABSTRACT	The purpose of this document is to establish the foundation elements of the PROMISE End-to-End Security architecture which will guide the detailed investigation and eventual specification of the security features of all layers of the PROMISE architecture.

STATUS OF DELIVERABLE		
ACTION	**BY**	**DATE** (dd.mm.yyyy)
SUBMITTED (author(s))	David Potter	12.12.2007
VU (WP Leader)	David Potter	12.12.2007
APPROVED (QIM)	Dimitris Kiritsis	15.12.2007

Revision History

Date (dd.mm.yyyy)	Version	Author	Comments
28.03.2007	0.1	David Potter	Initial draft
29.04.2007	0.2	David Potter	Some small additions incorporating feedback from HUT, InMediasP, SAP and Trackway
08.10.2007	0.3	Kary Främling	Changed the structure of the document, integrated new section 3 written by Kary Främling and the section "Potential attack scenarios" written by Katja Seidler. Cleaned up duplicated sections and text.
09.10.2007	0.4	Kary Främling	Wrote section 2 and transformed section 3 from bullet points into plain text
10.10.2007	0.5	Kary Främling	Finalised sections 2 and 3. Combined earlier sections 4 and 5 into one single and re-organised their contents
22.11.2007	0.6	Kary Främling	Reviewed the whole document, made some changes in sections 4 and 5 so that they would conform more with the rest.
28.11.2007	0.7	Kary Främling	Integrated Mario Neugebauer's document "WP_R13_SecurityArchitectureProposal.doc" into Section 5, made editing to make it fit in with the already existing text
06.12.2007	0.9	Hong-Hai Do	Review and change of Section 5 & 6
12.12.2007	1.0	David Potter	Final edit.

Author(s)' contact information

Name	Organisation	E-mail	Tel	Fax
David Potter	INDYON	david.potter@indyon.de	+44 23 9234 5152	+44 23 9259 2327
Kary Främling	HUT	Kary.Framling@hut.fi	+358 50 5980451	+358 9 451 3665
Katja Seidler	SAP	katja.seidler@sap.com		
Mario Neugebauer	SAP	mario.neugebauer@sap.com		
Hong Hai Do	SAP	hong-hai.do@sap.com		

Table of Contents

List of Figures

Abbreviations

Abbreviations used in this document:

BOL	Beginning of Life
DC	Device Controller (PROMISE Middleware)
DfX	Design for X
DHL	Device Handling Layer of the PROMISE Middleware
DSS	Decision Support System
ELV	EOL Vehicle
EOL	End of Life
GUI	Graphical User Interface
ISC	Inter System Communication
MOL	Middle of Life
PDKM	Product Data and Knowledge Management
PDM	Product Data Management
PEID	Product Embedded Information Device
PKI	Public Key Infrastructure
PLM	Product Life-cycle Management
PMI	PROMISE Messaging Interface
PROMISE	PROduct life cycle Management and Information tracking using Smart Embedded systems
RHL	Request Handling Layer of the PROMISE Middleware

1 Introduction

A sound and integrated security foundation for all levels of the PROMISE architecture will be a **Critical Success Factor** in determining the acceptance of PROMISE technologies in the marketplace. If PROMISE-compliant solutions are not secure, then acceptance will be severely restricted. Therefore we must approach the subject of security very seriously in order to protect the existing technical investment in PROMISE.

1.1 Dependencies

None of the PROMISE project demonstrators has any stated security requirement; therefore there is no absolute requirement to **implement** any aspect of security in PROMISE technologies before project M42 (May 2008). However, the PROMISE architecture and interface specifications must include a comprehensive, integrated, end-to-end security infrastructure by the end of the project.

1.2 Purpose of document

The purpose of this document is to establish the foundation elements of the PROMISE End-to-End Security architecture which will guide the detailed investigation and eventual specification of the security features of all layers of the PROMISE architecture.

The detailed specifications will result from the requirements analysis, design and specification which will be the subject of tasks TR13.1 through TR13.3. The results of the latter tasks will be accumulated and documented in DR13.1 (End-to-End Security Architecture) and in the reference volumes of the PROMISE Architecture Series.

1.3 Standards

The PROMISE security architecture should be based on and take advantage of both existing and emerging security standards and technologies. It is unlikely that the PROMISE project will identify new security requirements leading to the introduction of new or extension of existing security standards. However, we will as far as possible seek to exploit both existing and emerging security standards and technologies in innovative ways.

2 PROMISE Security Context

In PROMISE, the Internet is the main medium for communication between different information systems, no matter if they are Product Data and Knowledge Management (PDKM) systems, Decision Support Systems (DSS), Product Embedded Information Devices (PEID) or other. As illustrated by Figure 1, all these different information systems can be grouped together under the concept of a "**node**", whose internal implementation is irrelevant as long as it is capable of communicating over the Promise Messaging Interface (PMI).

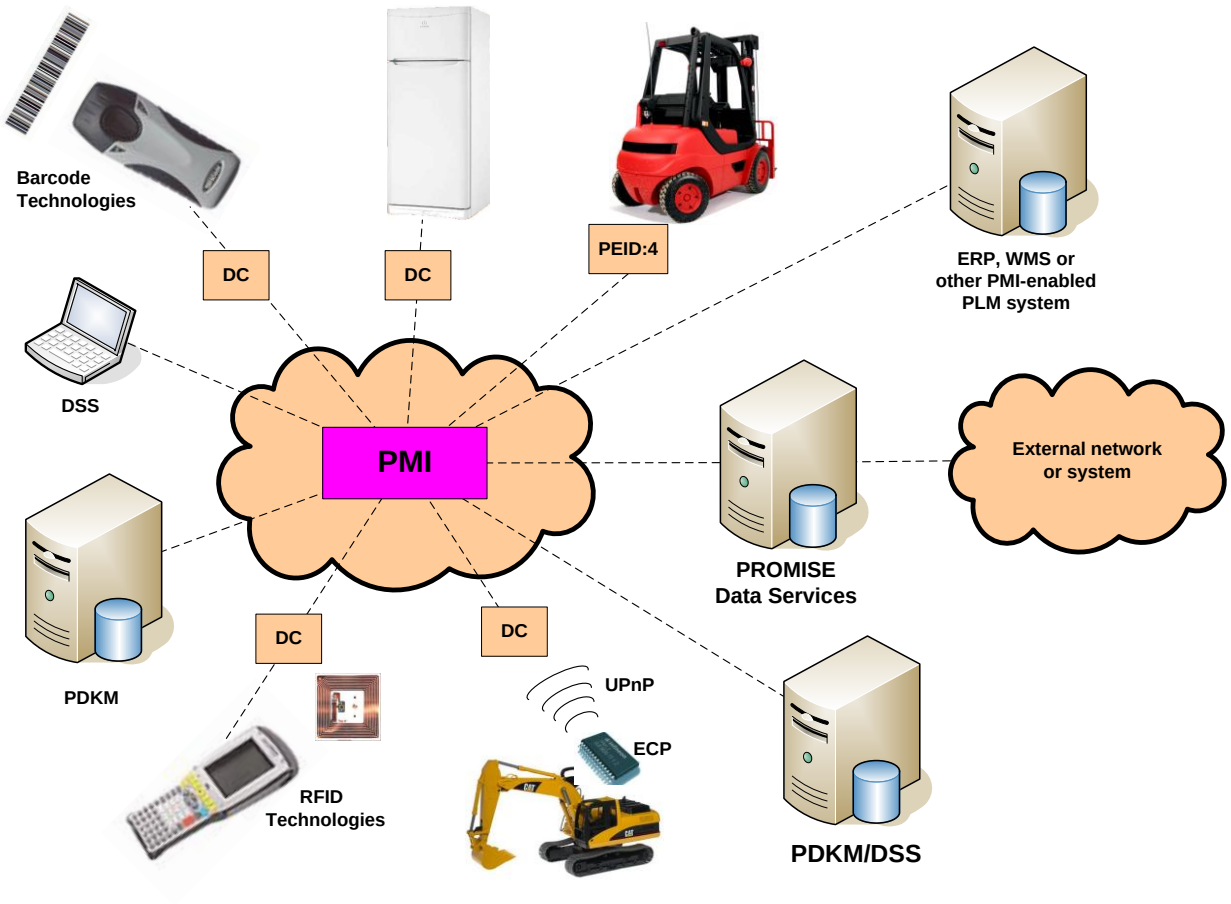

Figure 1. Illustration of PROMISE connectivity.

The PROMISE connectivity model is similar to that of the Internet itself. Where the Internet uses the *HTTP* protocol for transmitting *HTML*-coded information mainly intended for *human users*, PROMISE uses *PMI* for transmitting *XML*-coded information mainly intended for automatic processing by *information systems*. It is important to understand these relationships because PROMISE indeed proposes an extension to the Internet itself. This also signifies that the security architecture of PROMISE has to respond to the same security requirements as existing Internet-based services. One requirement is that a sufficiently high level of security should be possible to achieve when needed. Another requirement is that publicly available information should be accessible as easily as possible. An important requirement for the Internet is also that the privacy of its users should be respected so some level of anonymity can be required, which is typically in conflict with strict security requirements, where authentication of identities is crucial. These

different levels of security and their implications for PROMISE are analysed in detail in Section 3.

2.1 Handling of security for products with limited embedded computing power or network connectivity

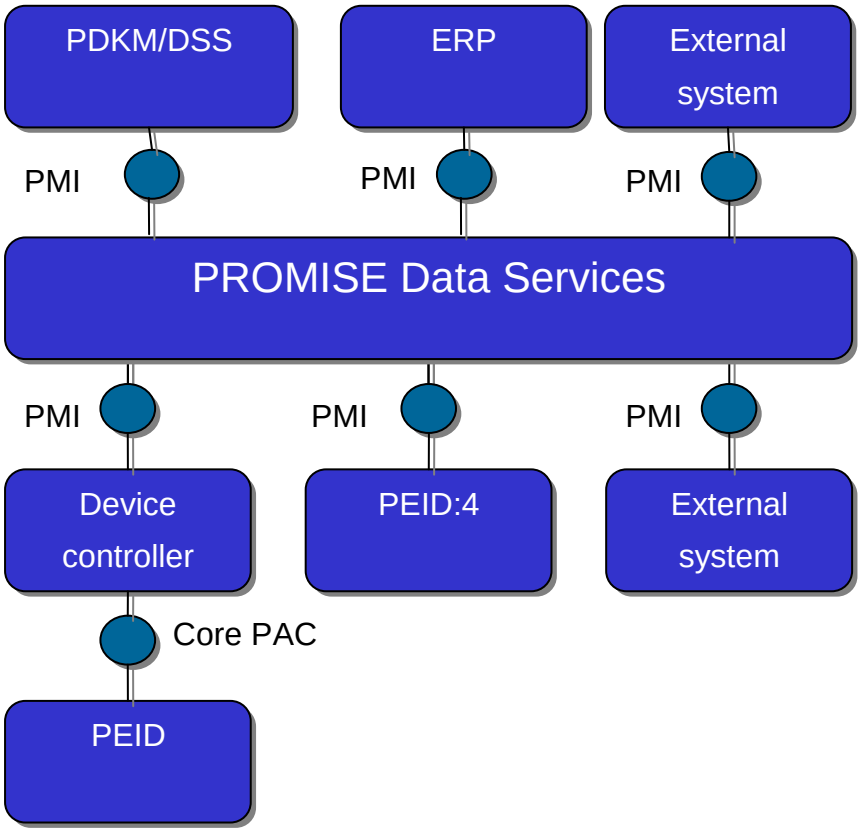

Figure 2. Use of Device Controller as PMI proxy for enabling PEIDs with limited computation power and/or network connectivity to access PROMISE Data Services, i.e. services provided by PMI-compliant nodes.

PROMISE security must be applied in a consistent and integrated way in all levels of the PROMISE architecture. If an embedded information device does not have sufficient computation capacity for implementing the PMI, then it can join the network through a proxy device. A grouping of such devices based on the amount of embedded computing power and network connectivity was defined in PROMISE deliverable "DR5.4: Generic PEID roadmap for each group", where the PEID:4 group was defined for devices who can support an embedded PMI implementation (or at least a partial PMI implementation). For all other PEID groups, the node consists of the PEID itself and a **Device Controller** (**DC**) as shown in Figure 2. The most universal DC implementation uses the Core PAC (Core Product Embedded Information Device Access Container) interface defined using the Universal Plug-and-Play (UPnP) protocol. In that case, the security mechanisms provided by UPnP can be applied to the Core PAC interface in the same way as the security mechanisms of the Internet can be applied to the PMI.

In applications where the Core PAC interface is replaced by some proprietary solution, i.e. a *proprietary DC*, the implementation of security is up to the proprietary DC and therefore outside the scope of PROMISE. A myriad of such PEIDs exist already, e.g. engine control units in cars give access to diagnostics and measurement information using the OBD-II or CAN protocols but

they do not allow modifying control parameters in a way that would endanger the correct operation of the car. Another example is television receivers, where parents can prevent their children form accessing certain channels and protect these settings with a password. The same receivers also have access control for accepting firmware updates. The PROMISE security model must take into consideration and support the use of such existing and future security implementations. In practice this means that the PMI must have support for passing such necessary proprietary security credentials when needed. Because any kind of information can be sent using the PMI, the sending of such security credentials (encrypted or not) is already supported.

2.2 Overview of security terminology and technologies

Security requirements can be roughly grouped together into the following categories:

1. *Encryption of stored data and network traffic*: This is necessary in order to avoid that sensitive data is accessed by unauthorized parties.
2. *Integrity check of data (stored and transmitted over network)*: Especially when data is transmitted over a network, it may be necessary to verify that none of it has been modified during the transmission. This is often used e.g. for verifying that software packages downloaded over Internet arrive as they should to their destination. Otherwise viruses or spying software could be introduced by some intermediate server during the transmission.
3. *Authentication*: In order to restrict access only to trusted parties, it is necessary to authenticate the computer and/or the user that attempts to connect over a network before providing access to information.
4. *Access control*: When a requesting computer and/or user has been authenticated, access rules can be defined for what information is accessible and in what ways, e.g. read, write, execute, delete etc.

Examples of corresponding solutions are:

1. *Asymmetric and symmetric encryption*. Asymmetric encryption uses one key to encrypt information and another to decrypt the information, usually called a public key and a private key. The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message encrypted with the public key can be decrypted only with the corresponding private key. Examples of widely used asymmetric encryption methods are RSA [Kaliski and Staddon, 1998] and the Digital Signature Algorithm (DSA) [NIST, 2000]. Symmetric encryption uses only one key that is kept secret. Because symmetric encryption is less computation-intensive, it is usually used for network transmission. In that case, a joint symmetric key is generated by one of the communicating parties and transmitted to the other parties using asymmetric encryption.
2. *MD5 (Message-Digest algorithm 5)* [Rivest, 1992] is often used for integrity check of e.g. un-encrypted software packages distributed as ISO CD-images or Zip files. If the data is encrypted, a successful decryption operation signifies that the message has not been modified so there is no need for a separate integrity check.
3. *Public Key Infrastructure (PKI)* is the most used authentication method for Internet services. It is based on the use of certificates according to the X.509 standard [Adams et al., 2005]. A certificate specifies e.g. who is the owner of the certificate, what organisation certifies the authenticity of the certificate, the asymmetric encryption method used for encrypting data with the certificate, a public key and validity dates. The PKI is a

hierarchical structure where the uppermost level consists of trusted certificate authorities such as VeriSign, Comodo etc., who sell authenticated certificates to other organisations. However, authentication may also be performed without using a PKI. In trusted networks, it may be cheaper and simpler to store public keys of trusted parties and use them directly for performing an authentication check as is done by the Secure Shell (SSH) system [Ylönen, 2006], for instance.
The most traditional authentication method are usernames and passwords, which are usually as valid as the ones mentioned above as long as they are stored and transmitted in encrypted format. Internet banking applications typically use this technique, where the PKI first certifies the bank's identity to the user and allows creating an encrypted communication channel using Secure Sockets Layer (SSL) [Dierks and Rescorla, 2006] encoding with the HTTPS protocol [Rescorla, 2000]. Fingerprints, iris scanning etc. are other variants of this authentication category.

4. *Access control* is based on authentication; once a computer and/or user has been authenticated using one of the methods mentioned above, then access rules can be applied. Access rules are stored in different ways depending on the information system. On a Unix server, they are mainly stored in the "/etc/passwd" file, a database system usually has a user table with corresponding access rules and an ERP system may have its own system. In the case of PEIDs, RFID tags exist that are e.g. read-only, write-once, readable and/or writable only to authorized parties etc. Other PEIDs such as car engine control units may implement other, manufacturer-specific access control systems. This diversity of proprietary existing and future access control systems demonstrates that there is no "one-fits-all" solution. Therefore PROMISE provides means to integrate with these diverse solutions when needed, rather than attempting to propose some new system.

This background information on the PROMISE security context should be sufficient for analysing and understanding different security requirements and potential solutions presented in the next section.

3 Analysis of representative use cases with different levels of security requirements

In many projects, security tends to be the system consideration that is handled the last. The reason for this could be that implementing security mechanisms even using standard technologies requires managing certificates, keys, access rights, firewalls etc., which may require a lot of manual work and the involvement of many different people. Many organisations do not even have any personnel who would know how to set up secure servers, manage certificates etc. In order to also allow such organisations to provide and use PLM-related services, it is important to adjust the level of security according to the requirements of the service, rather than always imposing the highest possible level of security. Furthermore, not only organisations may need to query and update product information during its lifetime as illustrated in Figure 3. At least when the users are individuals, extensive security requirements could discourage the use of many services. At the same time, privacy issues become relevant. As shown by the still ongoing debate about the use of RFID technology, universal identification of individuals is a controversial issue.

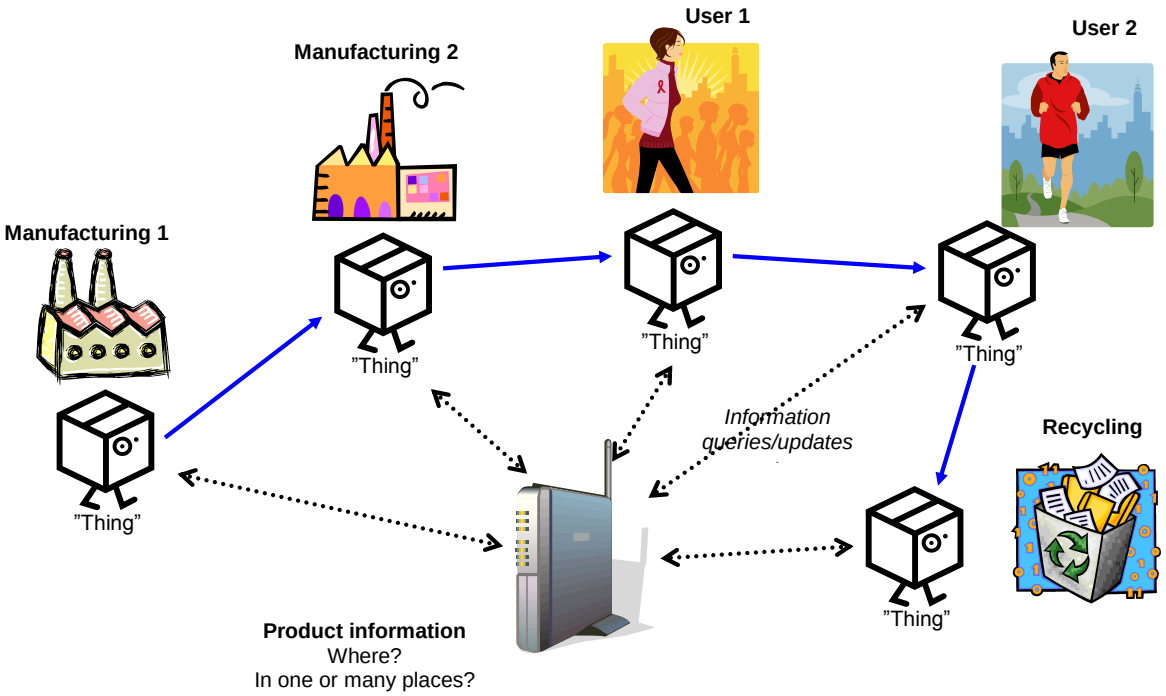

Figure 3: The product lifecycle seen from an Internet of Things point of view. Information about the "thing" is used and produced during all phases of its lifecycle [Främling and Holmström, 2006].

In the following sub-sections we will analyse cases where different levels of security are appropriate, beginning with no security requirements and finishing with the greatest security requirements.

3.1 No security requirements

Figure 3 illustrates the Internet of Things but if the "Thing" would be replaced with a computer screen and a web browser, it might as well illustrate the current Internet. In the current Internet, "query" and "update" operations are mainly performed using the HTTP protocol's GET and POST actions. In the PROMISE view of the Internet of Things, the PMI is used instead. Because PMI also has HTTP as the underlying protocol (augmented with SOAP), it could indeed be considered as an extension to HTTP. We will attempt to illustrate the case when no security is (and should be) required, both for the case of information queries and information updates.

Querying for information about product items could signify asking for user manuals, asking for online diagnosis of some trouble based on sensor values of the "Thing" (sent as parameters of the query) or a microwave oven that asks how long to heat the pizza that was inserted into it. These examples may not be relevant business cases as such but they do illustrate useful services where managing certificates or passwords would mainly be a burden.

Updating information could consist in reading a cinema publicity with a unique identifier that allows buying a ticket by reading the identifier with a mobile phone, registering for product update messages for the car that was just bought (simpler and more dynamic than the current system!), tracking of items in a closed environment, i.e. updating an item's location (e.g. for laboratory equipment). However, updating information usually requires a higher level of security than just querying for it.

The conclusion is that a great number of business scenarios exist where no security is needed and where introducing security would make systems more complicated, difficult to use and subject to

privacy issues. It is even probable that there are services that people would not use at all if they are required to identify themselves (i.e. authenticate themselves) unambiguously straight away.

3.2 Data encryption and integrity check is necessary

Encryption and data integrity checks become necessary in all applications where an external spy could gain access to confidential stored data or data sent as messages over a network. As pointed out in Section 2, data encryption usually also signifies data integrity check so that prevents an external party from tampering with the data. Data encryption can be achieved even when only one of the communicating parties has security credentials (typically a PKI certificate), as in Internet banking applications.

Some examples of situations when it is sufficient to authenticate only one party through certificates or similar are:

- When authentication is done by username/password or some other credentials as in Internet bank applications
- A serial number of an RFID tag, a vehicle identification number or similar existing identification methods can also be used; since encryption is activated in the beginning, none of this information is visible to external observers anyway

Data encryption and integrity check are technically quite simple to set up because existing protocols such as HTTPS (supported by all web servers) can be used directly. Even small companies might have the necessary know-how for this or can cheaply buy the setting up of it. Since each organisation only needs to manage its own certificate (i.e. just have it stored in the right place), there is no further maintenance effort needed. Because HTTPS is supported in all "client" applications such as web browsers, introducing this level of security doesn't introduce hardly any overhead. One advantage is also that the identity of the information provider is guaranteed, as well as the integrity of the information provided.

3.3 Authentication of all communicating parties is necessary

In the previous section, only one communicating party was authenticated by X.509 certificates or similar electronic authentication. In Internet banking applications, the authentication of the user is performed by usernames, passwords and/or some other means. In machine-to-machine communication it is often required that all parties would be electronically authenticated, i.e. that all parties want to verify the security credentials (X.509 certificates or similar) of the others through a "handshake" operation before proceeding with the communication. HTTPS supports this by a technique called "client authentication". When using client authentication, the web server takes care of that only authenticated parties can connect.

Despite the existence of client authentication, the technical challenges may increase considerably when using this level of security. Some reasons for this are:

- SSL client authentication is usually configured on the web server level, which implies some modification of configuration files by the web master. In small companies with skilled web masters, this will not be a problem but in bigger companies and/or with less skilled web masters it could be a major problem.
- The result of the authentication may not be accessible to the actual service (e.g. a PDKM) so it may be necessary to do the authentication again if needed e.g. for access control.
- If someone deletes or copies a certificate, then it could enable unauthorized access to information of many parties. Certificates can be revoked but it may be difficult to notice

that someone else is using the certificate so significant damage may occur before the certificate is revoked.

- For individual users, creating a certificate is still technically challenging and the user may not even want such a certificate to be transferred.

For Web Services using the Simple Object Access Protocol (SOAP) [W3C, 2007], an alternative to "client authentication" is to use the Web Services Security standard [Oasis, 2004]. The WS-Security standard specifies how to include security credentials in the header of SOAP messages, thereby allowing the services themselves to do the authentication. Various software products exist (including open-source products) that implement WS-Security. With WS-Security it is also relatively easy to encrypt parts of a message. Still, setting up WS-Security is technically even more challenging than setting up HTTPS client authentication.

As a conclusion, this level of security will probably be reserved for inter-organisational communication between organisations that have the means to set up the needed authentication mechanisms. However, as technology evolves, this level of security could become more feasible for other contexts also.

3.4 User-based access control is necessary

By the word "user" we will in this section normally assume that it is an individual user, an organisation or a computer that has been authenticated by one of the means described in the two previous sections. User-based access control is needed if the accessible information or services depends on who is trying to access them. As pointed out in Section 2, many systems already have in-built access control. Therefore the implementation of a "sufficient" degree of access control is rather a question of specification and technology choice of the application owner.

In current PROMISE application scenarios, access control is provided by mySAP on the PDKM level and by various proprietary solutions on the PEID level. On the PEID level, UPnP-based solutions that use the PROMISE Core PAC interface have a range of authentication and user access tools at their disposal. The conclusion is therefore that there should be no need for PROMISE to define specific access control methods or tools because such methods and tools already exist and can be used as such in PROMISE applications.

3.5 Parts of messages need to be encrypted

Transmitted messages could, at least in principle, contain pieces of data that are intended for different recipients. Therefore different parts of the message may be encrypted using different keys so that only the final authorized party may access it. In the current PMI specification, this is possible by including encrypted infoItem values that only the right recipient knows how to decrypt. This is again an application-specific issue that does not affect the PROMISE architecture or interfaces as such. There are also other technical solutions to implement partial encryption of messages such as using encrypted attachment files or functionality provide by WS-Security specification.

4 Security Management in PROMISE

The earlier sections have dealt with security on a rather generic level, which should be sufficient for dealing with the cases of "no security" and "data encryption without authentication". In this section we will analyse in further detail how the following cases can be handled in a PROMISE security system:

1. Authentication of users (human and system)
2. Data Security (Confidentiality and Authenticity)

PROMISE security must be applied in a consistent and integrated way in all levels of the PROMISE architecture. The diagram below shows the architecture layers in their simplest form.

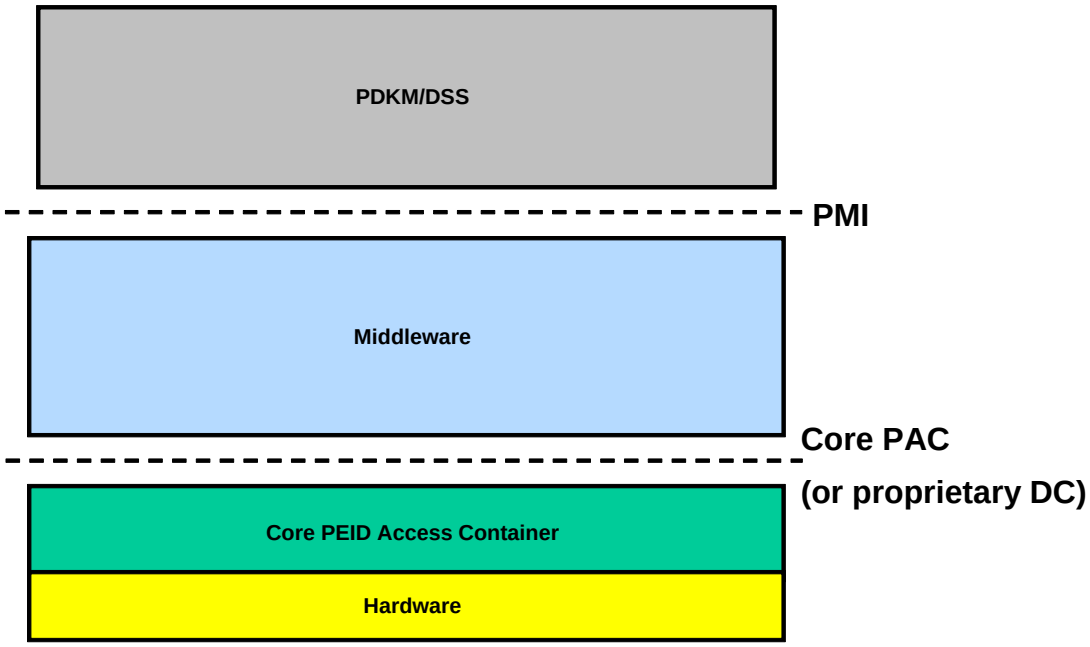

Figure 4: Illustration of basic components in the PROMISE Architecture. Other components are also possible as long as they are "PROMISE-compliant" by implementing the PMI or the Core PAC interfaces.

The diagram shows the three primary component layers of the PROMISE architecture, which comprises the following:

1. The PDKM/DSS: the integrated back-end PROMISE system comprises the PDKM database and the Decision Support System which use the PROMISE Messaging Interface (PMI) to connect to and make data requests via the PROMISE middleware layer. Any back-end system can potentially be enabled using the PMI to connect to the PROMISE middleware.
2. The PROMISE Middleware: the middleware provides communications and discovery services on behalf of its users (back-end systems and PEIDs), including device control functions for the management of PEID access.
3. The PEID Layer: comprises product identification devices which may range from very simple PEIDs (e.g. barcode and passive RFID tags) to high-function PEIDs such as on-board computers which may even imbed part of the middleware function.

The components in each of these architecture layers has a role to play in the end-to-end PROMISE security architecture. In a simple view, these roles range from defining and managing security policy at the back-end, through managing authentication of users and observing security attributes in the middleware, to storage of data with the correct security attributes on the PEID.

The complexity of these roles and their boundaries must be sufficiently flexible to take account of different levels of sophistication in the implementation of components and to allow for proxy implementations as necessary. Examples of different roles played by these components are:

- Back-end systems enabled by the PROMISE Messaging Interface (PMI). These may include not only a PROMISE PDKM/DSS but any 3rd-party back-end system enabled by the PMI, even distributed PROMISE system management functions (e.g. stand-alone PROMISE Metadata Manager, Connection Manager or Device Registry).
- Interconnected middleware instances which together form a PROMISE Inter System Communication (ISC) network. One middleware instance should be able to authenticate any neighbouring middleware instance when/if needed.
- Device Controllers (DCs) which manage the connection between certain levels of PEID and the PROMISE middleware
- Sophisticated PEIDs which can connect directly to the PROMISE middleware (i.e. the PEID software or firmware implements at least parts of the PMI).
- Simple PEIDs (e.g. RFID tags) attached to high-value products.

We will now analyse the roles of the different components more in detail.

4.1 Back-end Systems

In PROMISE, the main backend system is split into four components: core PDKM, PDKM GUIs, core DSS business logic and DSS GUIs. These components must not necessarily be running on one server. Hence, also the DSS should authenticate towards the PDKM as well as the GUIs should authenticate towards the DSS and PDKM respectively. Only permitted GUI instances should be able to access the DSS and PDKM content.

Back-end systems using the PMI, which can include PDKM and DSS instances, must provide adequate security checks and controls to ensure only authorised human users may access them and that those users' authority is controlled according to their user group permissions.

The mechanisms for such control should be aligned with the appropriate user access controls for the operating system type and underlying technology bases such as database, PLM system and user interface technologies. PROMISE back-end systems, such as the PDKM, must be able to authenticate the origin of requests for data or metadata.

In order to ensure security and integrity of data in PROMISE, we must have the flexibility to define, manage and conform to the attributes of any single data item at all levels of the PROMISE architecture.

A user of PROMISE infrastructure must be confident that the desired security of any single data item will be respected whether that data be stored within the PDKM/DSS, in transit or stored long or short-term anywhere within the PROMISE Middleware (including Device Controllers), and while stored on any PEID.

The desired security attributes of a single data item may differ according to the level of the PROMISE architecture, and the differences may not always be intuitive from a hierarchy standpoint. As an example, the on-board odometer value of a vehicle must obviously be writeable on-board the vehicle (PEID or On-Board Computer (OBC)). It may be permitted for an authorised service agent to re-write the value from an authorised Device Controller or distributed back-end system, but perhaps it should not be permitted to alter this value elsewhere in the middleware or other back-end system in order to prevent falsification.

Security attributes should allow among others:

- read-only data
- encrypted data

4.2 Middleware

For secure communications links we can consider mechanisms such as HTTPS and similar. For authentication of users connecting to the middleware, we can consider local authority definitions in .ini files, registries, symmetric/asymmetric key pairs, or more sophisticated controls using certificates or other trust-based approaches.

Any PROMISE Middleware implementation should support the following options:

- authentication of back-end systems (including PDKM/DSS) that attempt to connect to it
- authentication of neighbouring middleware instances in an ISC network
- authentication of Device Controllers (DCs) that attempt to connect to it
- authentication of PEIDs that have the capability to attempt a direct connection to it
- the ability to transmit and receive data over secure connections

The security of data that is being transmitted by the middleware, or being held temporarily in intermediate storage (e.g. store-and-forward nodes), must not be compromised.

4.3 Device Controller

When dealing with “simple” PEIDs that do not have any significant processing power, the Device Controller (DC) will have to operate as the proxy for managing security.

A PEID may contain secure data, which might for example be encrypted, but might also be “clear” data limited to be read only by users with appropriate permissions. The DC must respect such security attributes. It may be necessary for the DC to retrieve the metadata structure for a PEID from the PEID itself or a Metadata Manager in order to determine the correct attributes.

Similarly the DC must be responsible for ensuring that data is written to a PEID according to the security attributes defined in its corresponding metadata structure.

It may also be an advantage to be able to limit the scope of authority of certain Device Controllers, e.g. maybe certain Device Controllers are only permitted to read and never write data.

4.4 PEID

In the case of any PEID with built-in processing capability, the responsibility for compliance with authentication and data security attributes may be wholly supported by the PEID itself, or shared with a Device Controller according to the level of PEID sophistication.

PEID identification data may also include simple authentication mechanisms such as check digits verifiable by algorithms which apply to distinct product types or serial number ranges.

When simple PEIDs (e.g. RFID tag) are attached to high-value goods, it may be necessary to have a choice of appropriate mechanisms to authenticate the PEID itself in order to identify counterfeit PEIDs, and to be able to detect if any PEID has been substituted or tampered with in an unauthorised manner.

5 Potential attack/security scenarios

All devices and software systems involved in PROMISE application scenarios typically exchange confidential product information with each other. Hence, they are potential subjects to attacks by unauthorized parties, who want to gain access to such information or to manipulate the information exchange. This is aggravated for Internet-enabled systems such as PROMISE.

The goal of this section is to identify potential attacks, how they could be performed against PROMISE systems and how they might be avoided. The main protection goal is to ensure *confidentiality*, *integrity*, *availability* of, and finally, *access control* to confidential PLM-related information. In Sections 5.1 to 5.5 we will analyse all potential attack scenarios. In Section **Error! Reference source not found.** we will draw some conclusions on the severity of these scenarios and how they could be avoided.

5.1 Attack scenarios with malicious objects (masquerading)

A first attack possibility for an adversary is to introduce counterfeit devices or systems in any level of the PROMISE architecture.

1. Malicious PEID is introduced
An attacker may introduce a malicious device pretending the identity of an existing device. Once connected to the PROMISE environment, the device has all possibilities to send and receive data like the genuine device, making active eavesdropping possible. In particular, the malicious device can affect data stored in the PDKM (new wrong data could be introduced) and thus influence decisions made by the DSS. Furthermore, it is also able to receive certain data from above layers, such as PEID-specific operational data from the PDKM. Protection against such a device can be achieved by means of encryption and authentication of PEIDs.

Similarly, a malicious PEID can pretend a new identity when connecting to the DC. Doing so, the attacker may get some enterprise internal data via automated registration processes, if no secure authorization mechanisms are applied. This issue can be solved on the one side by only allowing previously registered devices to connect to DC. On the other side, security mechanisms can be developed, such as to assign access rights to granted users (PEIDs) and to verify authenticity of PEIDs, so that new PEIDs fulfilling security requirements can also connect to the PROMISE environment.

2. Malicious DC is introduced (applicable to distributed Middleware implementations with separate DC and RHL components)
A fake DC introduced by attackers may cause serious problems to PROMISE since may obtain and manipulate information from the PEIDs. A malicious DC is not necessarily stationary like in typical installation of PROMISE DCs, but it can deliberately move to find new PEIDs and collect/manipulate data from PEIDs found its range. Without proper authentication and authorization mechanisms, a PEID would not be able to distinguish between a genuine and a malicious DC.

In case that no PEIDs are found or connected, a faked DC may pretend the presence of connected devices. On the one side, it can request data from systems from the upper layers, like PDKM and DSS. On the other side, it can deliberately introduce wrong information when answering requests from such systems. Even in case that some PEIDs are available, a fake DC may hide all or some connected devices, raising the impression that they are not reachable. Therefore, requests to such devices cannot be served. More severely, such a DC may allow further malicious PEIDs to connect to the PROMISE environment.

In order to deal with these issues, authentication mechanisms need to be supported by the DC and Middleware, in case that they are connected through an unsecure network. A DC should prove its authenticity to the RHL in order to prevent malicious requests or responses. Similarly, a DC also needs prove its authenticity to the PEIDs so that the PEIDs can deny any requests in case of a suspicious DC.

3. Malicious RHL is introduced (applicable for Middleware implementations with separate RHL and DC components)

In case that a counterfeit RHL is introduced, we observe similar threats as in the case of a malicious DHL. In particular, such a RHL can provide PKDM and DSS with wrong information or gain illegal access to information in PDKM and DSS. It may pretend and/or manipulate the existence of DHLs and PEIDs, allow malicious instances of DHL and PEID to connect to the PROMISE environment. As a consequence, the RHL requires therefore the same security mechanisms as the DHL.

Authentication mechanisms between DC and RHL are required before communication between them can be established. A DC has to be able to identify a counterfeit RHL in order to reject any requests for data from it. It is also necessary to prevent DCs from sending information about the availability of PEIDs. Furtermore, the PDKM should also be able to identify any connected RHL in order to prevent sending requests to a malicious RHL.

4. Malicious PDKM is introduced

Considering that the PDKM is the central storage and management component for all product data, the threats associated with a malicious PDKM are extremely critical to the PROMISE application scenarios. A malicious PDKM can communicate with the Middleware and inquire data from all PEIDs connected to the PROMISE environment. Furthermore, it can manipulate the data obtained and serve the DSS with wrong information. Altering or deleting certain data on a PEID or in the PDKM will become critical in terms of ensuring data integrity and correct actions proposed by the DSS.

Using the PMI interface, a counterfeit PDKM can inquire data also from other PDKMs. Therefore, it is necessary for other systems in the PROMISE environment, in particular, DSS and the Middleware, and other PDKM instances connected through the PMI-network, to prove authenticity of any PDKM instance in the network. This requires authentication mechanisms between the corresponding systems.

5. Malicious DSS is introduced

A malicious DSS with access to the PDKM can have access to all confidential product information. Furthermore, it can manipulate decisions performed on the data or deliberately derive wrong decisions which can influence the PLM processes supported in the PROMISE environment.

In order to cope with these issues, authentication mechanisms are necessary between DSS and PDKM. They should ensure that only DSS which are able to authenticate are allowed to read data from the PDKM and to derive business decisions from this data.

6. PEID is accessed by an adversary (e.g. stolen)

As PEIDs are attached to products, they can be stolen by attackers. This represents a source for serious threats that need to be considered. In particular, a number of activities can be performed by an adversary with a stolen PEID. First, all current operational data stored on the PEID is read out, if it is not encrypted. If not protected appropriately, the attacker can obtain the encryption/decryption mechanism implemented on the PEID and apply them to eavesdrop messages to and from this type of this devices. Ultimately, the attacker may replicate the device by means of reverse engineering in order to use the copy devices for malicious attacks (as

described in 1. In order to deal with these issues, corresponding encryption and protection mechanisms are required for PEIDs. Alternatively, PEIDs may be chosen and installed in such a way that it cannot be stolen and/or tempered with easily.

5.2 Attack scenarios with eavesdropping

A second attack possibility for an adversary could be the eavesdropping of the communication between components of the PROMISE architecture. This is possible at any interface between two system components in the PROMISE environment: such as, betwen RFID-Tag and RFID Reader (PEID), between PEID and DC, between DC and RHL, between RHL and PDKM, between PDKM and DSS, and third-party systems.

Eavesdropping can be performed either passively or actively:

- *Passive eavesdropping:* Passive eavesdropping cannot be detected or avoided. The adversary merely needs to be in the range of wireless communication devices or has to attach himself to applied network infrastructure.
- *Active eavesdropping:* Active attacks are mainly so called man-in-the-middle attacks. These attacks also require access to the communication infrastructure, like the passive attempt. In contrast to a), the attacker acts as unrecognized intermediate between two communication partners. Communication party A and B assume a direct communication to each other, yet the attacker is the actual communication partner of A and B. He now can collect, alter, or delete arbitrarily information he relays.

The following eavesdropping scenarios are possible between PROMISE components:

- *Between PEID and DC:* It is possible to eavesdrop information which the PEID sends to the DC. So it is possible to retain PEID specific information.
- *Between DC and RHL:* Requests to and answers from DC as well as information about availability of PEIDs could be eavesdropped.
- *Between PMI-enabled RHL:* Requests to and answers from RHL could be eavesdropped.
- *Between PDKM and DSS:* Requests to and answers from PDKM could be eavesdropped.
- *Between PDKM and PDKM:* Requests to and answers from PDKM could be eavesdropped.
- *Between PDKM and Third-Party System:* Information from PDKM could be eavesdropped, when a user accesses them over a distributed system.
- *Between DSS and Third-Party System:* Decisions from DSS could be eavesdropped, so that user without access to DSS could get the information as well.

In order to protect the PROMISE environment against eavesdropping, following solutions can be considered:

- *Encryption*: Passive listening on communication channels cannot be avoided. Encryption mechanisms are thus necessary in order to prevent the communicated information to be understood by the adversary. Different security levels may be introduced in order to support encrypted transmission of data. In cases, such as, simple PEIDs (tags) without own encryption functionality, the components in the upper layers of the PROMISE architectures, e.g., DC, RHL, need to provide corresponding protection mechanisms.
- *Authentication*: Both the source and destination of each communication should have to verify their real identity using corresponding authentication mechanisms to avoid man-in-the-middle attacks. This has been discussed for the attack scenarios with malicious objects (masquerade).

5.3 Attack scenarios with authorization

In a distributed environment like PROMISE, authorization is required in order to prevent misuse of the systems. Unfortunately, it is also a critical security factor of the whole environment, as most attacks, like password probing, phishing, aim at gaining illegal access through security holes in authorization. To protect against such attacks, it is essential to define and implement a clear model of authorization and access rights.

For each system and user participating, it is necessary to decide which rights (concerning access to functionalities and to data) can be applied. Not every user or system should be allowed to request special data or manipulate it. Some instances should even only be allowed to read data in the PDKM. Access-levels are required to avoid false instances or components accessing critical data. Currently the PDKM already supports sophisticated authorization mechanisms so that the threats from this kind of attacks are less critical.

5.4 Attack scenarios with data loss

Another attack possibility for an adversary is to manipulate or destroy special components. For example a database or a part of it could be destroyed or manipulated in such a way, that the data is useless for any other participants. Another example is that a PEID with captured is destroyed or stolen. In such cases, it is important for the rest of the PROMISE environment to have the possibility to recover the data which was lost or at least to recover to the last save point. Backup and recovery functionalities are already supported by the PDKM for its data. However, such functionalities may also be required by other systems, in case this kind of attacks can occur with them.

5.5 Denial of Service and Buffer Overflow attack scenarios

A Denial of Service(DoS)-Attack aims at blocking the whole system so that it cannot be used. In an internet environment are such attacks always possible, and thus representing a high security threat for PROMISE. With DoS-Attacks, it is possible to obstruct nearly every PEID or software component, like Middleware, PDKM, and DSS, in the PROMISE network. Attackers could send requests (connection or query requests) to devices in short intervals and block regular requests and communication that are required by the business participants in the PROMISE environment. Although a DoS attack can be easily detected (by monitoring communication and responses in the network), it is difficult to generally secure the systems in a public network from DoS-attacks. Using authorization and/or specific network protocols, the systems can establish a private network with each other so that the threat of DoS attacks can be reduced.

In connection with the UPnP-protocol, Buffer Overflow-attacks also represent a large security problem to the PROMISE environment. In particular, malware could be introduced into the system using Buffer Overflow attacks and the attackers could gain complete control of the device (maybe also to the whole network). The UPnP-protocol is used between the PEID and the DHL, so these devices need special protection against buffer overflow-attacks. More importantly, each program-part should be checked for potential buffer overflows in order to reduce the risk of this kind of attacks.

5.6 Security mechanisms required by PROMISE

In order to illustrate the required security mechanisms, we take example of the FIAT application scenario in PROMISE and discuss in the following the different security scenarios and the

utilization of the corresponding mechanisms in order to handle the security issues. The use case scenarios are presented in Figure 5 and Figure 6, respectively, and are discussed in the following.

Tom is employee in repair shop ‘RS1’, where car ‘C’, which has never been before in RS1, is brought to for a normal check. Tom will drive car C into the repair shop to get information about special devices of the car. The information is stored on several PEIDs in Car C. Before Tom could get the stored information, the DC, which is normally located in the repair shop, has to realize the new PEIDs which had been brought into reach of it.

Due to the fact, that C has never been in RS1 before the DC doesn’t “recognize” the PEIDs, but of course could see some basic facts about them. The new PEIDs have to be logged on with the help of Tom. The system will ask Tom, whether the new PEID should be registered. To be sure, that Tom has the right to make this decision, he has to login before to authenticate himself. This security mechanism is quite important, due to the fact that otherwise everyone could register new malicious PEIDs without problem.

If car A will be brought to the repair shop RS1 again after this first check, the DC will recognize the PEIDs in the car as familiar ones. The registration process could be skipped over. Independent of this an authentication process must be passed to avoid that malicious devices could intrude into the Promise system.

If Tom wants to get information about the state of the car he will ask the PDKM or DSS-device over the computer system of FIAT. It is supposed that the PDKM is stored globally in the fleet management of FIAT, which is not based in the repair shop RS1. Tom’s request is sent over the intranet to the fleet management of FIAT, where the PDKM/DSS will manage the request and return an answer. The transmission over the intranet should be secured as well. Otherwise eavesdropping becomes possible. As mentioned before Tom has to be logged in to execute requests and make decisions. Not each employee should have the same rights concerning the data, which could be requested and sent. Therefore different access rights should be granted to the employees.

The PDKM forwards the request to the repair shop, in case of requests needing current data. The transfer is handled over the intranet. Encryption is needed to secure the transmission between different devices. It is conveyable that the encryption is handled as an end-to-end-connection from PEID to PDKM/DSS over Middleware, due to the fact that the Middleware does not have to deal with the data stored on the PEID. If an end-to-end-connection is used there would be no chance to eavesdrop the transmission in between. The encrypted data therefore must be checked against falsification.

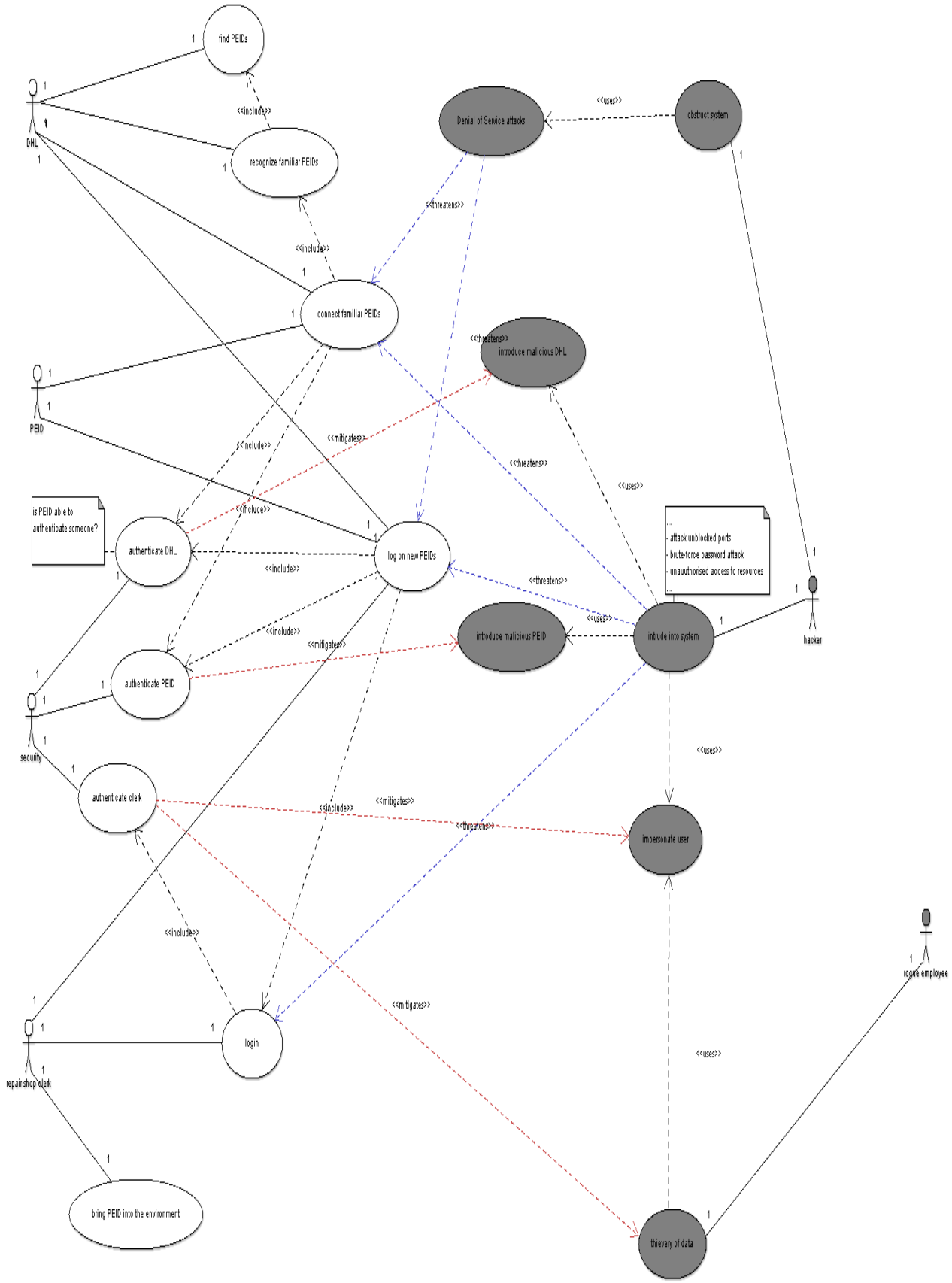

Figure 5. Use case diagram for security issues in Fiat scenario (1)

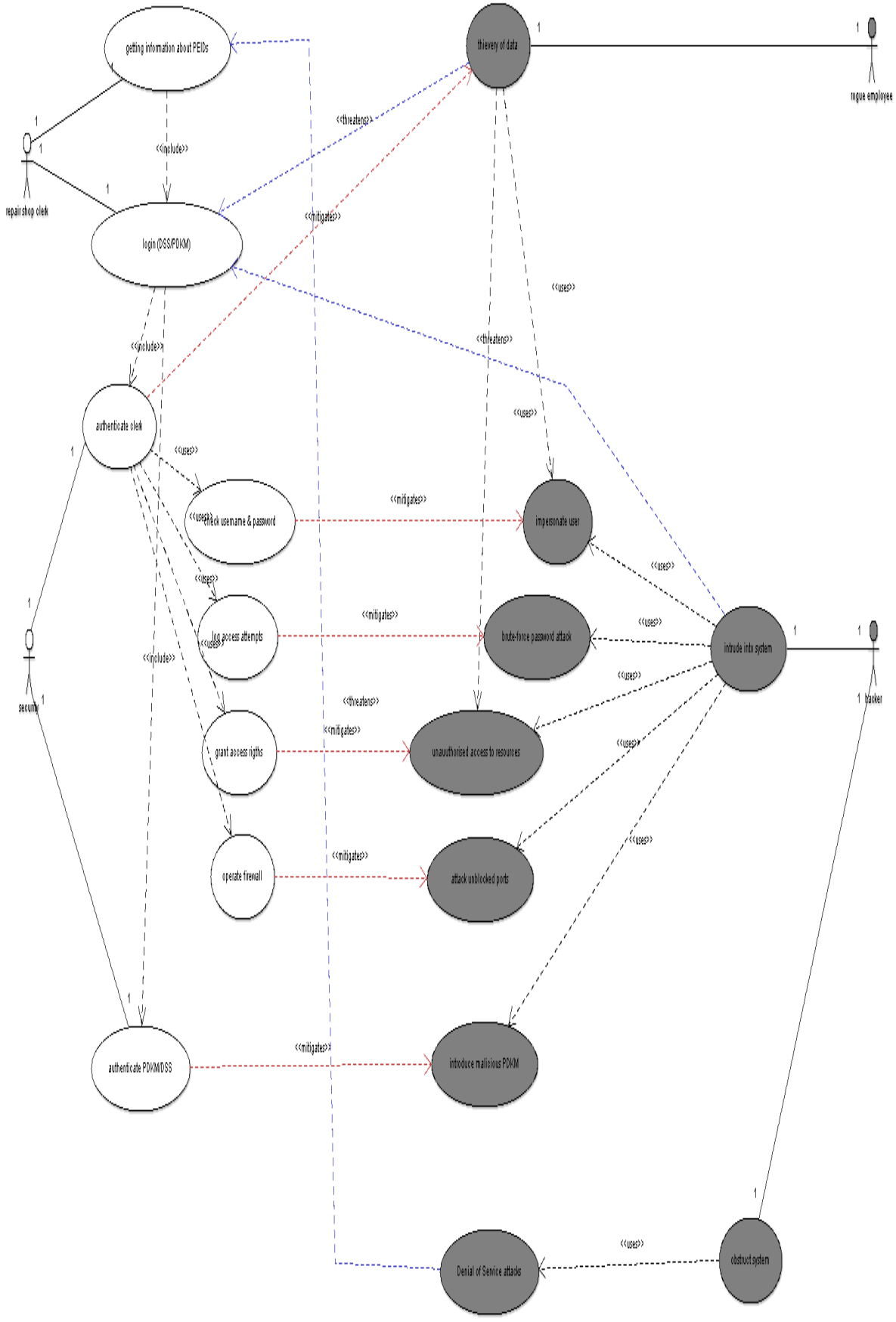

Figure 6. Use case diagram for security issues in Fiat scenario (2)

To summarize the discussed security scenarios and necessary security mechanisms, **Error! Reference source not found.** give an overview of the entire PROMISE architecture with its components PEID, DHL/RHL (Middleware), PDKM, DSS, as well as the users and other third-party systems. Furthermore, the figure indicates the necessary security mechanisms that are needed in order to make the communication (requests and responses) between the components secure: encryption of data and authentication between components. Built on this insights, the next section will describe concrete propositions in order to implement the security mechanisms in the interfaces between the PROMISE components

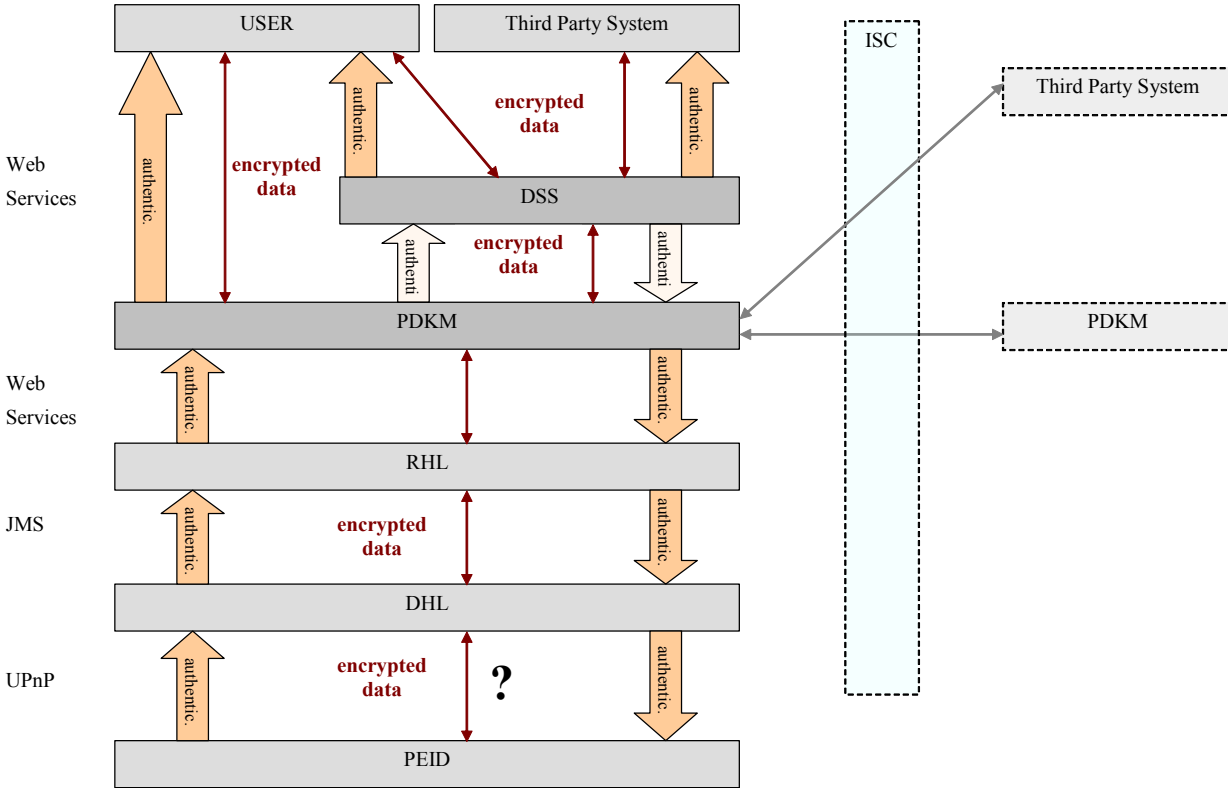

Figure 7. Necessary security mechanisms in PROMISE architecture

6 Concepts and interfaces for handling security scenarios

Based on the preceding analysis the major security issue in applications with high security requirements is that only granted PEIDs should be allowed to communicate to the PROMISE middleware. The introduction of malicious MW will be assumed as less important. This can be avoided e.g. by permanent connection of the MW to the PDKM (for instance during installation of PROMISE infrastructure). Hence, the following three main challenges for handling of the dynamic components (PEIDs) in the PROMISE infrastructure remain:

- Initialization of the PEID/PDKM key pair
- Introduction of malicious PEID

In the following two subsections the two processes for the initialization of the PEID and the reconnection to the PEID to the PDKM are explained. Inherent in these descriptions the means to avoid attacks from malicious PEIDs or PDKM are contained. We do not deal with details of symmetric (e.g. DES, 3DES, AES) or asymmetric (e.g. RSA) encryption. Rather, an idea for

securing a PROMISE installation will be presented. For concrete implementations it will be required to examine the encryption standards in detail and determine an approach that fits best for the PROMISE requirements.

6.1 Initialization of the PEID/PDKM key pair

A new product part appearing in the PROMISE system will have a non-initialized PEID attached to it. On the one hand this means that the PEID is not yet associated to the particular PROMISE security concept. On the other hand, there is no representation of the part in the PDKM. This means that the new PEID first must be associated with the actual PROMISE world. Hence, the process serves the initialization of a PEID.

The initialization phase is supported by the prototypical Process GUI which is employed in particular in the A2 demonstrator. It is deployed within the PDKM system and closely couples with the core PDKM. The aim of the Process GUI is to initialize product parts which are new in the PROMISE system. Additionally, the backend of the Process GUI will support the initialization regarding the security.

For the initialization it is assumed that the PROMISE middleware, PDKM and PEID are trustworthy. This can be assured since the middleware, PDKM and PEID work in the own infrastructure for sure. The idea is to issue a dedicated security key to each PEID. It will serve the secure data exchange between the PROMISE infrastructure and trusted PEIDs.

The initialization of the PEID regarding the security key for later communication with the PROMISE infrastructure will proceed as follows:

1. A trustworthy PEID connects to the PROMISE middleware.
2. The Process GUI shows the new product part which is to initialize.
3. The Process GUI initiates the key generation for the specific PEID with an individual, possibly non-unique ID. In parallel a unique ID for the PEID in the PDKM is created.
4. The PDKM invokes the method in the middleware to set the key and the unique ID for the new product part.
5. The middleware forwards the key and ID setting to the PEID.

As a result the key/ID pair is set in the PEID and in the stored in the PDKM for later access of the PEID to the PROMISE infrastructure.

6.2 Reconnection of the PEID to the PROMISE infrastructure

After initializing the PEID the associated product will go into its real world life and possibly operating data will be recorded. Eventually, the product will enter a service station for maintenance or exchange of parts. At this stage, it must be assured that at the one hand only initialized PEIDs can connect to the PROMISE infrastructure and that on the other hand only authorized PROMISE middleware can access data from the PEID.

Assuming the situation that a product with an attached PEID enters the service station, the following steps will be made to integrate a properly initialized PEID:

1. PROMISE middleware detects an appearing PEID with its public ID.
2. PROMISE middleware requests PDKM to generate a challenge message for the appeared PEID with the certain public ID.
3. PDKM creates a challenge message and transmits it to the PEID via the PROMISE middleware.
4. PEID must compute a response and sends it to the PDKM via the middleware.

5. The middleware checks the response for correctness and grants access to the infrastructure when the response is correct. Otherwise the PDKM instructs the middleware to cancel the connection since the connection to a malicious PEID should be avoided.
6. Attached to the grant-message from the PDKM to the middleware an answer is delivered to the PEID.
7. The PEID checks the answer from the PDKM and holds the connection if the answer matches the PEID's response. Otherwise the PEID disconnects, assuming a malicious middleware trying to connect to the PEID.

6.3 Connections between the static components

Beside the connection of the dynamic PEIDs to the PROMISE infrastructure also the static components (Device Controller, middleware, PDKM) require a secure interconnection. Therewith, it must be avoided to expose internal information (e. g. operating data from a PEID) on the way in the PROMISE infrastructure (e. g. from middleware to PDKM).

PROMISE components which are to be employed in real world scenarios are to install in an enterprise infrastructure before usage. During this system integration phase the PROMISE infrastructure will be installed and configured. Configuration regarding the security means that the trusted components of the infrastructure are to connect to each other. In particular the connection between the PROMISE middleware and the PDKM will be secure after installation and configuration.

Different means based on standard technology are available. For example, SSL is a common technique to secure communication between partners such that the content is remains hidden for external parties. Additionally, a firewall can be used to restrict the access to the PDKM only from associated PROMISE middleware components. Concrete means to deploy in a scenario are to elaborate during the planning for the integration. It mainly depends on the existing system architecture, IT-guidelines and best practices in the organization. Additional concepts or interfaces are not required so secure the communication between the middleware and the PDKM.

If the DSS is deployed separated from the PDKM the communication between them needs to be secured as well. Hence, the concept of standard technologies applies in this situation as well.

6.4 Proposal of Interfaces

6.4.1 Interfaces for Initialization

In Figure 8, a message sequence chart for the successful initialization is depicted. Therein, the following methods play the key role.

6.4.2 generateKey in PDKM

The generateKey method is offered by the PDKM and will be called from the middleware or the Process GUI to start the initialization. The generateKey function will accept a public PEID-ID. The generated key will be stored in the PDKM as associated key to a PEID and delivered to the middleware for initialization of the PEID.

6.4.3 setKey

Setting the generated key in the PEID will be supported by the method setKey. It is offered by the middleware and the PEID. The method is called with two paramters. One for the public PEID-ID and the other for the key which will be associated with the PEID.

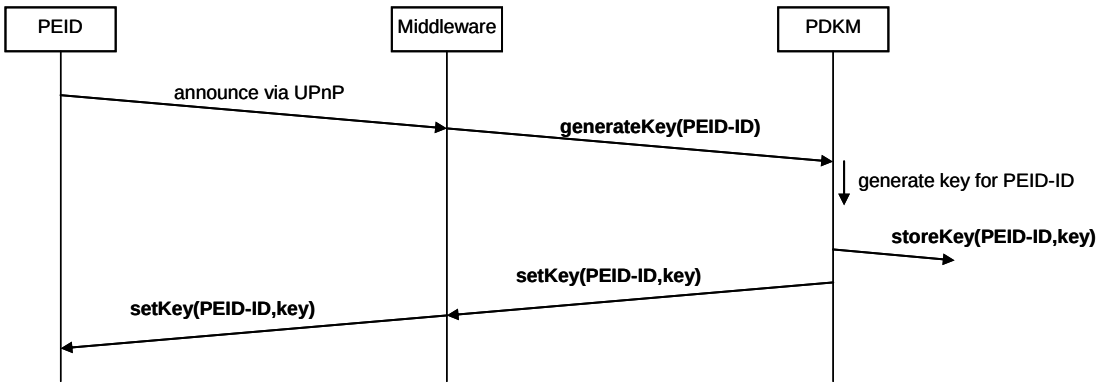

Figure 8. generateKey and setKey for the security initialization in the PROMISE infrastructure

6.4.4 Interfaces for Reconnection

Figure 9 shows the operation of the components during reconnection. In the following the interfaces serving to secure the connection are described in detail.

6.4.5 createChallenge

An appearing PEID will announce itself with the UPnP protocol to the middleware. The middleware then has to check whether the PEID is allowed to connect the infrastructure. Therefore, the middleware issues the request createChallenge to the PDKM.

6.4.6 askForChallenge / confirmChallenge

After requesting the PDKM with createChallenge, the PDKM will present the request to the service technician. The service technician then has to accept the request for challenge individually for each PEID. Thereby, the service technician can detect malicious PEIDs aiming to receive multiple challenges from the PDKM for key extraction. The confirmation from the service technician will be fed back into the PDKM and the registration will be continued or cancelled.

6.4.7 challenge

The challenge method is called by the PDKM in the middleware and by the middleware in the PEID, respectively. A challenge message is transported as a parameter in the method. A PEID receiving the challenge will have to generate a matching response.

6.4.8 response

After generating the response the PEID forwards the response message to the middleware and further to the PDKM. The response message is taken as a parameter in the response message. A PDKM receiving a response will first check if it matches with the issued challenge. A successful matching response will then initiate a confirmation with an answer message to the middleware.

6.4.9 confirm

The answer from the PDKM is sent via the confirm method to the middleware and finally to the PEID. A matching answer approaching the PEID will initiate the PEID to hold the connection to the middleware.

6.4.10 cancelConnection

If the response received in the PDKM does not match the initial challenge, the PDKM will issue cancelConnection which will result in disconnecting the PEID.

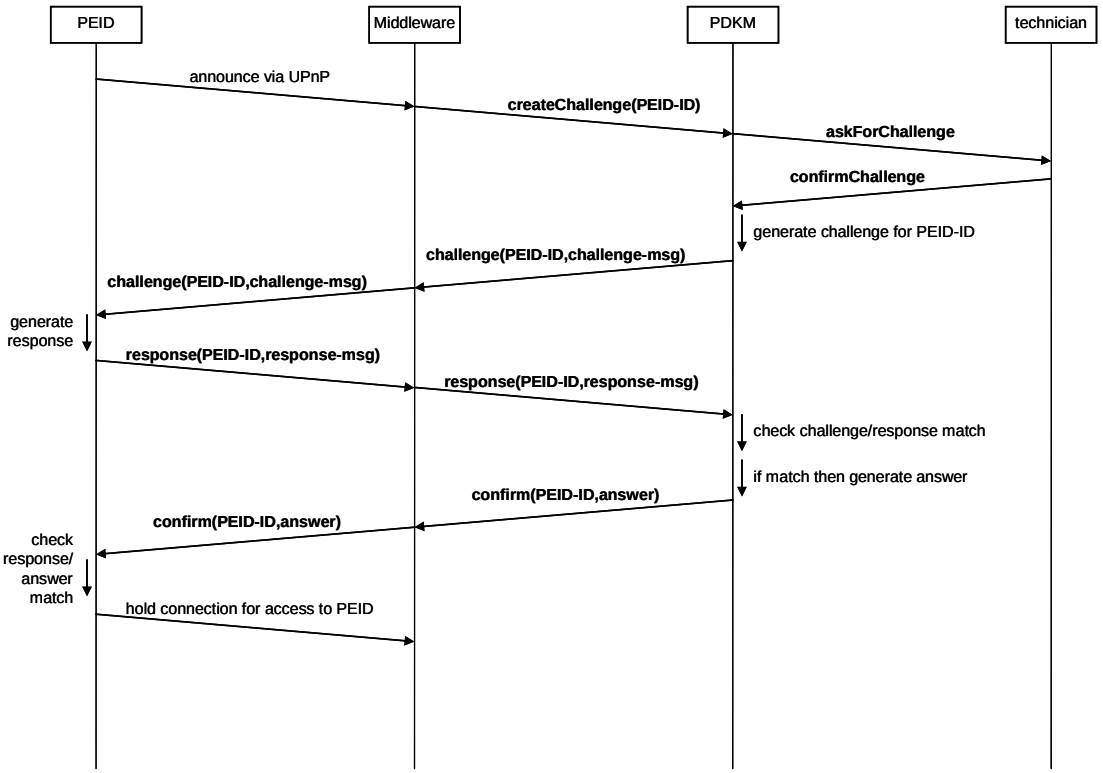

Figure 9. Operation during reconnection

7 Conclusions

A comprehensive and effective security architecture is a fundamental requirement for a complete PROMISE infrastructure. Without a consistent end-to-end security management concept, the acceptance of PROMISE technologies and solutions will be limited. The proposed end-to-end security management concept also allows for "no" security in applications where e.g. systematic authentication is not needed or desired but where additional security levels can be added later if and when needed. This flexibility is important from the point of view of making all kinds of usage scenarios possible and to keep implementation and management of security feasible according to application requirements.

This paper has analysed and defined the basics of a PROMISE security architecture that is mainly based on existing standard protocols and technologies. The conclusion is that a full end-to-end PROMISE security implementation is possible to implement without new standards or protocols. As such implementations are realised, new requirements may obviously occur but those requirements will most probably not be specific to PROMISE so corresponding solutions are likely to be developed by organisations such as IETF, W3C or similar.

8 References

Adams, C., Farrell, S., Kause, T., Mononen, T. (2005). *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP) – RFC4210*. Available online (accessed 10 October, 2007): http://tools.ietf.org/html/rfc4210

Dierks, T., Rescorla, E. (2006). *The Transport Layer Security (TLS) Protocol - RFC 4346*. Available online (accessed 10 October, 2007): http://www.ietf.org/rfc/rfc4346.txt

Främling, Kary, Holmström, Jan (2006). How to create evolving information models by a layered information architecture. In: *Proceedings of The Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP'2006)*, 11-12 September 2006, Budapest, Hungary. pp.173-178.

Kaliski, B., Staddon, J. (1998). *PKCS #1: RSA Cryptography Specifications, Version 2.0 – RFC 2437*. Available online (accessed 10 October, 2007): http://www.ietf.org/rfc/rfc2437.txt

NIST (2000). *Digital Signature Standard (DSS) – FIPS 186-2*. Available online (accessed 10 October, 2007): http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

OASIS (2004*). Web Services Security: SOAP Message Security 1.0*. Available online (accessed 10 October, 2007): http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Rescorla, Eric (2000). *HTTP Over TLS – RFC 2818*. Available online (accessed 10 October, 2007): http://www.ietf.org/rfc/rfc2818.txt

Rivest, R (1992). *The MD5 Message-Digest Algorithm – RFC1321*. Available online (accessed 10 October, 2007): http://tools.ietf.org/html/rfc1321

Ylönen, Tatu (2006). *The Secure Shell (SSH) Protocol Architecture – RFC4251*. Available online (accessed 10 October, 2007): http://www.ietf.org/rfc/rfc4251.txt

W3C (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Available online (accessed 10 October, 2007): http://www.w3.org/TR/soap12-part1/

