SicAri – A security architecture and its tools for ubiquitous Internet usage

---

Deliverable PF5

Enforcement of Security Policies within the SicAri-Platform

Version 0.9, 22.02.2006

Andreas Heinemann, TUD-TK
Jan Oetting, usd.de
Jan Peters, Fraunhofer IGD
Roland Rieke, Fraunhofer SIT
Taufiq Rochaeli, TUD-SEC
Markus Ruppert, Flexsecure
Björn Steinemann, Fraunhofer SIT
Ruben Wolf, Fraunhofer SIT

Document Information

Corresponding document version on CVS:

```
$Id: pf5-policies.tex,v 1.16 2006/02/22 11:44:06 rwolf Exp $
```

Author of this version:

```
$Author: rwolf $
```

Date of this version:

```
$Date: 2006/02/22 11:44:06 $
```

Contents

1 Introduction

The specification of the SicAri platform as described in Deliverable *PF3 – Specification of the SicAri Architecture* [12] already introduced a few security providing components. This document will provide SicAri's security policy integration concept. It will describe the components of the SicAri policy framework and their interactions in order to guarantee that all security relevant processes in the platform are fulfilled according to the underlying security policy.

Our security policy integration concept is based on the manifold requirements with respect to policies as described in Deliverables *Pol1 – Requirements on SicAri Security Policies* [16] and *PF2 – Requirements for the SicAri Architecture* [13]. Selected requirements are for example

- Control of all security related processes and tasks. Impossibility to bypass the policy enforcement component.
- Compatibility of the policy framework with the SicAri platform's plug-in approach. No need to change existing or upcoming services in order to enforce the platform's security policy. Transparency of policy control.
- Consideration of trade-off between expressiveness and complexity of the policy description language.
- Support for platform administrators during policy management.

Taking these requirements into account, we developed a policy framework for the SicAri platform.

The remainder of this document is structured as follows: Section 2 lists some aspects of the platform specification that need to be adapted in order to integrate security policies. Sections 3 and 4 cover identity management and authentication aspects. The components of the policy framework are outlined in Section 5, while Sections 6 to 9 describe particular aspects of the policy framework, such as policy enforcement, decision, provisioning, and generation. Finally, Section 10 gives a summary and an outlook.

2 Architecture Specification Version 2

This section describes changes and adjustments of SicAri's platform architecture specification as described in Deliverable *PF3 – Specification of the SicAri Architecture* [12] that have been necessary in order to integrate new functionality such as security policies into the platform. The references given in the description below refer to the respective sections and pages in Deliverable PF3. Specific parts of Deliverable PF3 are refined or completely replaced by this document, the relevant parts are denoted in the description below.

Section 4.2 – SicAri Kernel: Each entity working with the SicAri platform needs to be authenticated before starting operation. A proved identity is the basis for all further processes of an entity; in particular, access control decisions are made on the basis of a proved identity. Authentication is implemented by a Platform Authentication Service, which uses the platforms Identity Manager to verify the entity's credentials. Details about the various authentication mechanisms are given in Section 4 of this documents.

Section 4.2.2 – Environment: There is an additional proxy, the Security Proxy, which is responsible for automatic and transparent enforcement of the security policy. Details about the new Security Proxy are given in Section 6 of this document.

Section 4.2.3 – Security Context: The cooperation between the Security Proxy, the SicAri Security Manager and the Security Context is illustrated in Section 6 of this document.

Figure 5 on Page 17: As a start, the Context Manager will be replaced by a Context Database whose application is optional and required only for a very special type of security policies (such as the specification of history based access permissions). Likewise, the automatic Feedback Module is skipped in the current version of the platform architecture.

Figure 6 on Page 18: Policy Query and Policy Provisioning are realised using the Common Open Policy Service (COPS) Protocol. Details about policy provisioning are given in Section 8 of this document. Policy Specification uses the policy description language XACML. An outline of the policy generation process is given in Section 9 of this document. The full description can be found in Deliverable Pol4 [17].

Section 4.3.3 – Role-Based Access Control: As a start, the platform uses a simple role activation algorithm, since there is currently no need to activate and deactivate roles during a session. The algorithm may change if new requirements for security policies arise. In the current version of the role activation algorithm is as follows: All roles that a user may potentially have are activated at the time of the users login.

The least privilege principle and separation of duty concepts cannot the realised in the current version of the platform, since the actual version of XACML's role-based profile doesn't support it.

Section 4.3.4 – Context-dependent Access Control: The current version of the platform will only consider static context information within the access control decision. Sensors are currently not supported.

Figure 9 on Page 22: The Reference Monitor is realised by the Security Proxy (as a trigger for the subsequent policy enforcement). Details about the policy enforcement approach are given in Section 6 of this document.

Section 5.1 – Policy Enforcement ans Security Manager: The paragraph about the Context Manager on page 26 is obsolete.

Use Case "Access Control Enforcement": The user's identity in step 4 is part of the Security Context and doesn't need to be retrieved from the Identity Manager.

Figure 12 on Page 28: Context Manager is obsolete. The chain of execution is as follows: Service → Security Proxy → Security Manager → Policy Service (incl. Policy Decision component).

Section 5.3 – Security Provisioning: The current platform architecture is restricted to the enforcement of access control policies. There is no enforcement for security mechanisms.

Section 5.4 – Context Manager: This section is obsolete.

Section 5.5 – Authentication Manager: The platform supports various authentication methods such as username and password, or certificate-based authentication. Details about the

responsibilities of the Authentication Manager and its relations to the Identity Manager are given in Section 4 of this document.

Section 5.6 – Identity Manager: The responsibilities of the Identity Manager and its interaction with the Out-Of-The-Box-PKI (OOTB-PKI) are described in Sections 3 and 4 of this document.

Section 5.8 – Persistency Service: It turned out to be that the current platform architecture doesn't require a generic persistency component.

Section 6.1 – Sensor Modules: Due to the reduced functionality of the context component, there is no need for sensor modules.

Section 6.2 – Cryptographic Primitives: All new cryptographic primitives will be integrated using Java's crypto provider approach.

Section 6.3 – Communication Protocols: Various protocols (such as LDAP, SSL, COPS) have already been integrated.

Section 7 – Logical View: Since the SicAri prototype improves and evolves continually, the class diagrams may also be a target for changes.

Section 7.2.1 – Scenario: Check Permission and Sequence Diagram in Figure 30: Due to minor reorganisation and renaming of components the scenario and the illustrating sequence diagram will marginally change. The updated illustration will be given in Section 6 of this document.

Section 8 – Deployment View: Access to remote SicAri services is realised by the platform's web service framework which allows transparent access to locally available or remote SicAri services.

Details about integration of security policies into the SicAri platform and the interactions with identity management and authentication components are given in the subsequent sections.

3 SicAri Identity Management

Identity management comprises the administration of identities within a defined SicAri infrastructure as well as the provision of the *Identity Manager* service, which can be used during runtime by other services and applications on platform instances. Figure 1 depicts the different components involved.

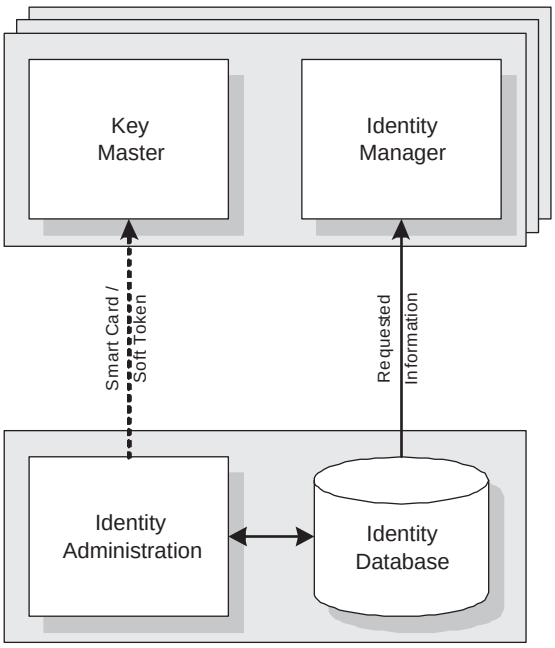

Figure 1: Components of the SicAri Platform for Identity Manangement

3.1 Identity Administration

Identity management in SicAri includes the *administration* of identities within a defined SicAri infrastructure. That is the registration of new identities (*rollout*), the modification of identity attributes as well as the revocation of identities. Identities and corresponding attributes are administrated globally and stored in the identity database within the SicAri infrastructure. It is planned to use well tested modules of the Flexi-PKI [5] to fullfill these administration processes including the generation of cryptographic key material and rollout of smartcards as security tokens. Since the results of the different processes are mapped into a LDAP database, this LDAP database is used during runtime as identity database within SicAri. Furthermore, the generated cryptographic keys and personalized smartcards conform to security standards. With this approach it is assured that the administration processes can easily be exchanged as long as the results conform to the same LDAP schema.

The following three types of identities will be provided in SicAri, whereas they only differ in the semantic meaning and the associated attributes. Syntactically all identities are stored within the same database using the same representation schema. Furthermore, *user* is used as synonym for *identity* throughout this document.

Platform administrators Every platform instance in the SicAri infrastructure is started by the associated *platform administrator*. The cryptographic keys of the platform administrator are used to unequivocally identify the platform instances during inter-platform communication processes. Thereby, mapping between platforms instance and platform administrators has to be one-to-one.

Service users To narrow down the set of permissions granted to basic services (cf. Section 6), these services are not initiated by the platform administrator itself during the bootstrap-

ping process of a platform instance, but by a correspondig *service user*. Thus, a service user is defined for a specific type of service and a well defined set of permissions is associated with this identity resp. with a specific role which is in turn associated with this identity. To support single-sign-own mechanisms a platform administrators is allowed to impersonate a set of service users without another authentication (cf. Section 4).

Regular users In general a *regular user* in SicAri corresponds to real person, who interacts with services of the SicAri infrastructure through local platform instances. Depending on the set of permissions granted to this user, he is either restricted to the use of the given service infrastructure, or allowed to administrate basic or application services provided.

One part of the user rollout is the generation and storage of cryptographic keys and corresponding certificates for authentication, digital signatures, and data encryption. In case of platform administrators, these keys are provided as *soft tokens*, i.e. a password-protected keystore file which is usually stored on the platform host. Furthermore, it is recommended to generate cryptographic keys for regular users. These keys are either stored on a personalized *smart card* or provided as soft token. In either case, a password has to be associated with regular users, which is used as fallback for authentication (cf. Section 4). Since a service user is an abstract identity, no cryptographic keys are associated with this user type. All certificates additionally stored in the identity database.

As mentioned above the identity database maps identities to a certain set of attributes. The following list comprises oblicatory identity attributes in SicAri, whereas this list can be extended depending on the current needs of provided services:

dname The distinguished name [9] unequivocally identifies the user in a hierarchical namespace. Furthermore, this distinguished name is used to identify a leaf in a LDAP database, and is embedded in the subject and issuer field of a X509 [10] certificate.

userid Another unequivocal identification of a user in the SicAri infrastructure (usually a short human readable name).

email The email adress of the user, used to send personal messages (in case of a regular user) or status information (in case of a platform administrator). As the *dname* or *userid* this attribute is globally unique within the SicAri infrastructure.

password The password hash which can be used together with the userid for user authentication.

certificate Up to three X509 certificates for authentication, signature, and encryption.

serialnumber Up to three serial numbers of the corresponding X509 certificates

In table 1 the mapping between SicAri user attributes and the attributes of the defined LDAP schema is given. The table contains further optional attributes which are conceivable in typical application scenarios.

The distinction of the three user types should rather be done by choosing the hierarchy layout (i.e. the structure of the distinguished name) than by defining additional attributes. Examples for distinguished names are shown in table 2.

SicAri user attribute	LDAP attribute	must/optional
dname	dn	must
userid	uid	must
title	title	optional
fullname	cn	optional
givenname	givenname	optional
surname	surname	optional
street	postaladdress	optional
postalcode	postalcode	optional
city	l	optional
country	c	optional
telefon	telephonenumber	optional
fax	faxsimiletelephonenumber	optional
mobile	moible	optional
email	mail	must
department	departmentnumber	optional
building	buildingname	optional
room	rommnumber	optional
photo	jpegphoto	optional
birthday	-	optional
password	userpassword	must
biofinger	biofinger	optional
certificate	usercertificate	must
serialnumber	serialnumber	must

Table 1: Mapping between SicAri Attributes and LDAP Attributes

3.2 Identity Manager

In contrast to identity administration as described above, the *identity manager* service does not modify the identity database. Its task is the provision of identity information to other services and applications. Thus, the identity manager is installed on each platform instance as basic service, which subsequently accesses the global identity database upon request.

The tasks of the identity manager service can be subdivided into:

search for identities Search for identities based on given user attributes: A set of `userids` is returned as globally unique identifiers for matching user records in the identity database.

mapping of identity representations Map the three globally unique user attributes `dname`, `userid`, and `email` into each other.

request of identity attributes Request user attributes corresponding to a given `userid` as unique user identifier.

As certificates are defined user attributes, these can be requested through the identity manager service. Thus, public keys to verify authentication credentials, encrypt data, or validate digital signatures of a specific user are available. As the name implies, the private key of a user's

Plattform administrator:	`emailaddress=platform1@sicari.de,`
	`cn=platform1.sicari.de,`
	`ou=Administrators,`
	`o=SicAri,`
	`dc=sicari,dc=de`

Service user:	`cn=Database Service,`
	`ou=Services,`
	`o=SicAri,`
	`dc=sicari,dc=de`

Regular user:	`emailaddress=jan.peters@sicari.de,`
	`cn=Jan Peters,`
	`ou=Users,`
	`o=SicAri,`
	`dc=sicari,dc=de`

Table 2: Example Distinguished Names for the three User Types

cryptographic key pair should never be revealed to the public, it is stored in the password-protected soft token or the cryptographic smart card.

To enable the basic cryptographic mechanisms user authentication, signature generation, and data decryption another service – the *key master* service (cf. Figure 1) – is provided on each platform instance. In case of a soft token, the private keys are temporarily loaded into memory to execute the cryptographic operation. Although the private key never leaves the local platform, the user has to trust the local platform instance, that is the local platform administrator. In SicAri, the smart card provides the highest level of security with respect to exposure of a user's private keys, as the private key never leaves the smart card. In this case, the key master returns the PKCS11 [18] provider as wrapper to cryptographic operations executed directly on the user's smart card. Furthermore, these operations must be approved by the user, which has to enter the smart card's PIN code on the card terminal everytime such an operation is performed. Since many users can concurrently use a local platform instance, the key master manages access to private keys – directly or through the smart card – for all users currently logged in. Thereby, access to keys is only granted to the corresponding user and as stated above to the platform instance itself.

Another functionality of the key master is the provision of a set of *trusted certificates* stored in the soft token of the platform administrator. These certificates – in particular the certificate of the root certificate authority (Root-CA) of the SicAri infrastructure – are used to validate certificate chains in the context of user authentication.

4 Authentication of Platform Entities

This section will be provided in the next version of this document.

5 Security Policy Integration Concept

5.1 Policy-related Components of the SicAri Platform

The policy integration concept of the SicAri platform requires the interaction of various components of the SicAri platform (see Figure 2). This section gives an overview of all the components involved and their interactions with other components. After a short outline of each component below, we will provide details about selected aspects in the Sections 6 to 9.

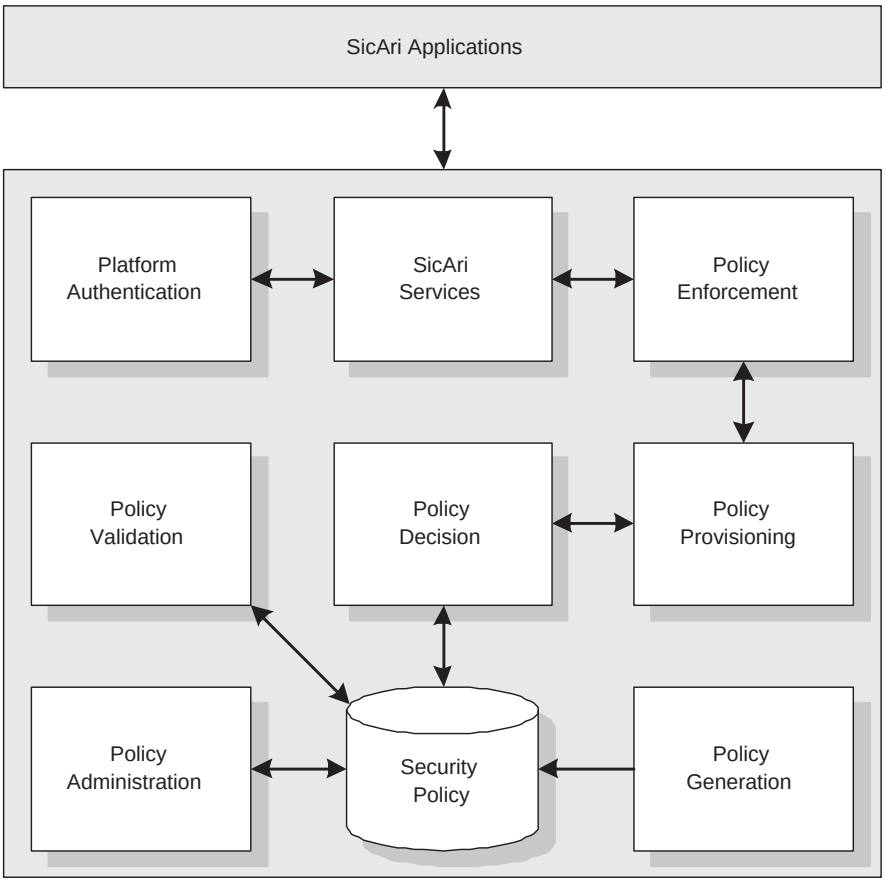

Figure 2: Policy Components of the SicAri Platform

SicAri Applications and SicAri Services. SicAri applications are located on the top of the SicAri platform. They interact with users of the SicAri platform on the one hand, and with SicAri services in order to provide their services on the other hand.

The SicAri platform hosts two kinds of services: basic services and application services (cf. Deliverable *PF3 – Specification of the SicAri Architecture* [12]). Both kinds of services are controlled by the platform's security policy enforcement components (see below).

The policy processing is mostly transparent. That is, SicAri services are not aware of the existence of the security policy. Therefore, there is no need to modify and adapt existing or upcoming services to be compatible with policy integration concept. From the time when a service is

registered in the SicAri platform as a SicAri service, it is controlled by SicAri's policy framework. The only thing that needs to be done is to configure the security policy for the new service. After that, the policy is enforced automatically.

Platform Authentication. There is a clear separation in the SicAri platform between authentication of platform entities and access authorization of platform entities to platform resources. Authentication provides a basis for subsequent access control decisions. As described in Section 4, the SicAri platform provides various authentication mechanisms.

Policy Enforcement assures that all security relevant tasks can only be fulfilled if they are in accordance with the underlying security policy. The policy enforcement component detects security relevant tasks, consults the policy decision component in order to decide upon a task, and enforces the policy decisions, i.e. allows a platform entity to access a platform resource or not. Details of the policy enforcement component are described in Section 6.

Policy Provisioning covers the distribution of policies, policy updates, policy decision requests and responses. Details about this component are given in Section 8.

Policy Decision. The platform supports two different usage scenarios for policy decision, that can be selected depending on the actual deployment infrastructure of the platform. Both scenarios are illustrated in Section 7.

Security Policy and Policy Generation. SicAri uses the widely spread standard *Extensible Access Control Markup Language (XACML)* as policy description language. The policy itself is generated using so called policy patterns that allow to specify template policies archetypes for recurring areas of application. A summary of policy patterns and the policy generation component is given in Section 9, a detailed specification can be found in Deliverables *Pol3 – Policy Patterns and Its Application* [15] and *Pol4 – Policy Generator* [17].

Policy Validation. The specification of the security goals does not necessary correspond with its implementation or realisation as specified in the security policy specification (e.g. the XACML policy file). In most cases security policies are administrated manually, that is mistakes may arise. It is necessary to evaluate, whether the security target set is actually implemented in the security policy specification. The SicAri platform provides a component that allows specification of security goals and checking whether they are met or not. Details can be found in Deliverable *PE5 – Evaluierung von Sicherheitszielen auf Basis von Policies* [14].

Policy Administration. Even if the SicAri platform provides the ability of automated security policy generation, there may be the need of fine granular policy administration, e.g. a new user needs to be added, or the permissions of a user or role need to be changed. For that, the SicAri platform will provide an administration API and a corresponding graphical user interface. This is the objective of upcoming Deliverable PF8.

5.2 Bootstrapping

This section will be provided in the next version of this document.

6 Policy Enforcement

This section will be provided in the next version of this document.

7 Policy Decision

After Section 6 focused on policy enforcement, this section covers how policy decisions are made within the SicAri platform. In principle, there are two different usage scenarios for policy decisions: (1) local and (2) remote policy decisions.

7.1 Policy Decision Scenario 1

The main characteristics of this scenario are the local *Policy Enforcement Point (PEP)* and the *Local Policy Decision Point (LPDP)*. The scenario is depicted in Figure 3.

The PEP interacts with the local policy service, which mainly consists of the following components: LPDP, cached policy, and COPS adapter. The LPDP is responsible for making policy decisions based on the input from the SicAri security manager and a locally cached version of the master security policy. The LPDP is realised by an extended version of Sun's XACML reference implementation (see below). The PEP uses the Java-API of Sun's XACML engine in order to communicate with the LPDP. A (potentially remote) policy provisioning component provides a copy of the latest master policy to the PEP-LPDP component using the *Common Open Policy Service (COPS)* protocol. The policy itself is specified using the XACML policy description language.

This is the favoured policy decision usage scenario for the SicAri platform. However, there is an alternate policy decision approach.

7.2 Policy Decision Outsourcing Scenario

The main characteristic of the second scenario is that a local PEP requests a policy decision from a remote *Policy Decision Point (PDP)*.

Here, the policy service mainly consists of a COPS adapter which transforms the policy decision request of the SicAri security manager into an XACML policy request. The COPS adapter sends this request to the remote PDP which is responsible for providing the policy decision based on the master security policy. The XACML policy decision response is sent back from the remote PDP via COPS to the local PEP which enforces the policy decision.

As in scenario 1, policy decisions are made by an extended version of Sun's XACML reference implementation. In this scenario, there is no locally cached version of the security policy. However, an enhanced version of the SicAri architecture may have the ability to cache prior policy decisions. In contrast to scenario 1, in which the whole XACML security policy is transferred using the COPS protocol, scenario 2 only transfers XACML requests and responses over COPS.

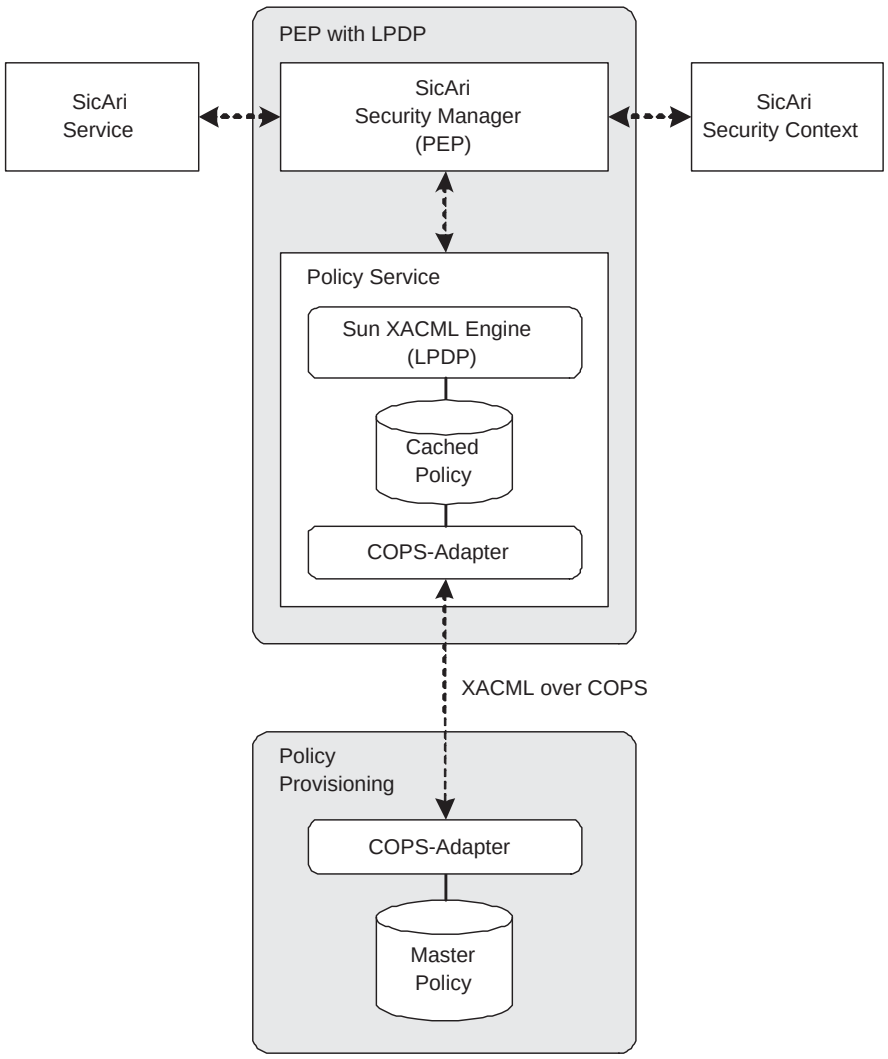

Figure 3: Policy Decision – Usage Scenario 1

7.3 Transport of Policy Decisions with COPS

Figure 4 gives an overview of the interplay between the different components in SicAri that execute policy requests and decisions. In this section we focus on the role that the COPS adapter components play in this process.

The SicAri COPS extensions are build on the COPS implementation from the Network and Systems Management group of the University of Waterloo. This software is written in Java and published under the terms of the GNU General Public License.

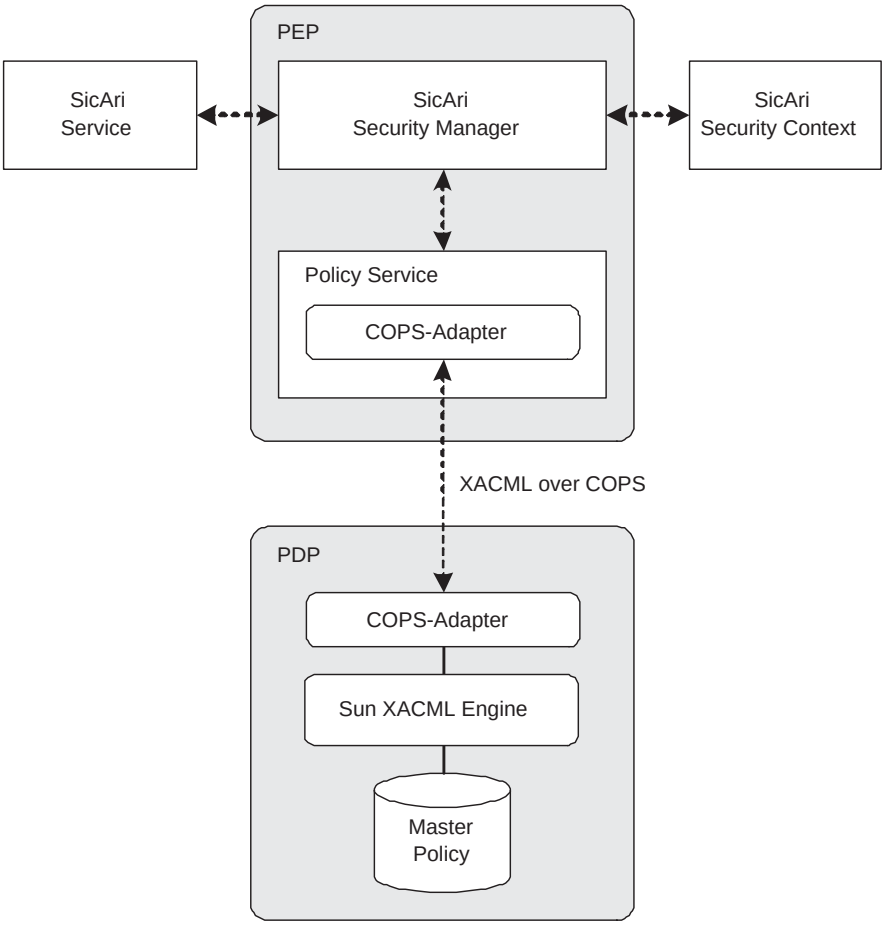

Figure 4: Policy Decision – Usage Scenario 2

7.4 Using Sun XACML Implementation to Evaluate RBAC Profile of XACML

Sun's XACML implementation [1] is an open source reference implementation of a XACML evaluator. It comprises of library classes which could be used in building Policy Enforcement Point or Policy Decision Point [**?**]. Currently, Sun's XACML implemention (Version 1.2) supports the complete mandatory specification of XACML 2.0 enriched with a number of optional features.

One of the main benefits of Sun's XACML implementation is the modular architecture, which allows the extension of the implementation. This ranges from adding new attribute types, which are user defined data types in XACML, to adding new finder modules, which search the policies according to the user's search criteria.

Our interest lies in using Sun's XACML implementation as a XACML policy evaluator, which is specified in RBAC profile 2.0. This is possible because the Sun's XACML implementation provides the extension of modules. We will sketch out the evaluation process of Sun's XACML implementation in order to explain the relationship between RBAC profile 2.0 and the modules.

The RBAC profile specifies the following three types of XACML policies in order to support core and hierarchical roles:

- Role Policy Set
- Permission Policy Set
- Role Authorization

The policy evaluation process can be roughly sketched as follows: at the time of policy enforcement, the XACML implementation (will be referred to as *evaluator* hereafter) receives the request in form of XACML context. This XACML request provides the information about the *target*, which acts as search mask for policies. This target consists of the information about subject, resource, action and environment. Using this target, the evaluator finds a policy which applies to this context, by comparing the target of context with the target of policies. After that, the evaluator evaluates the policy.

There are two modules, which play an important role in finding the policies matching the search criteria: `PolicyFinderModule` and `AttributeFinderModule`. The `PolicyFinderModule` searches for `PolicySet`, `Policy` or `Rule`, whose target matches the target of the request. Furthermore, it also searches for policies, which are referenced by `PolicySet` or `Policy`.

If the target in policies specifies only the attributes of subject or resource, instead of their unique identity, then the `AttributeFinderModule` should help the `PolicyFinderModule` to resolve all attributes of the subject or resource. By resolving the attributes of the subject or resource, a comparison between the targets could be performed.

In case of the RBAC profile of XACML, the *Role Policy Sets (RPS)* will be evaluated first. The `PolicyFinderModule` searches through all RPS, whose target matches the request target. Because the request's target contains the subject identity and the RPS' target contains only the role as subject's attribute, so a match could not be made. In this situation, the `AttributeFinderModule` should resolve the subject's attribute, which contains the subject's role. These attributes are defined in the role authorization file.

After the match candidates are found, the `PolicyFinderModule` continues to search other policies, which are referenced by the RPS. These referenced policies constitute the *Permission Policy Sets (PPS)*, which contains the permissions for each role. Note that, if these PPS do not have any policies with the intended target, but the PPS have references to other PPS, then the `PolicyFinderModule` should recursively follows these references, until a matching policy is found or all PPS in the references are evaluated. In the latter case, the evaluator should send an `NotApplicable` message.

As we can see, in order to enable evaluation of RBAC profile of XACML, one shall only need to extends these two modules. The `PolicyFinderModule` should be able to follow all policy references, and the `AttributeFinderModule` should be able to resolve the attribute defining the role.

8 Policy Provisioning

In the course of the SicAri project the Policies working group and the Plattform working group have been discussing different approaches how to provide the security policies to the SicAri nodes. It has early been decided to use the protocol framework for *Policy Based Network Management (PBN)* which has been defined by the *IETF Resource Allocation Protocol (RAP) WG*. The core of this framework is the *Common Open Policies Service (COPS)* Protocol [3]. The COPS protocol provides a means to communicate policies and policy decisions in a distributed system. The main characteristics are

- logical and architectual separation of policy enforcement and policy decision
- client/server model of PEP and PDP
- reliable transport of messages between PEP and PDP via TCP
- flexible and extensible through self-identifying protocol objects that allow to define arbitrary protocol payload
- stateful communication between PEP and PDP which share request/decision states that allows the PDP to asynchronously update decisions and configuration information at the PEP

COPS is designed to be used in two basic scenarios — outsourcing and configuration. In the outsourcing scenario the PEP delegates all policy decisions to the PDP what has already been described in 7.2. The PEP does not make any local decisions but may reuse decisions that it has received earlier and that have not been revoked by the PDP. This scenario is extremely useful when the mobile node where the PEP resides does not have the computational power or the memory size to handle local policies and decision processes.

If the PEP has a local PDP (LPDP) at its disposal it may rather opt for the configuration scenario. In this case the PEP sends a configuration request to the PDP and asks for the policies for those modules that it is responsible for. The PDP in turn answers with a stream of policy configuration data that will be used by the LPDP. For each chunk of configuration data that the PEP receives it sends a confirmation message back. This way the PDP keeps track of the information the PEP mirrors. Later on the PDP can provide update messages to the PEP if portions of the policy

become obsolete or are replaced with newer instructions. Figure 5 provides a schematic view on the interaction between the PEP and the PDP and LPDP.

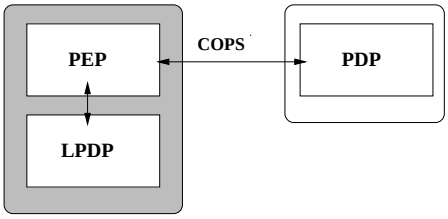

Figure 5: Policy configuration of PEP with LPDP

It has also been considered to employ the COPS-PR variant of COPS for policy distribution. The appendage PR stands for policy provisioning and extends the COPS protocol. Basically COPS-PR can be used whenever you would apply COPS in the configuration mode. The authors of this protocol extention motivate their work in the section *"Why COPS for provisioning?"* in [4] with

> *[. . .] COPS-PR allows for efficient transport of attributes, large atomic transactions of data, and efficient and flexible error reporting*
> *[. . .] it is defined as a real-time event-driven communications mechanism, never requiring polling between the PEP and PDP.*

COPS-PR transports policy data and assumes a named data structure that is known as the Policy Information Base (PIB). PEP and PDP share the same knowledge about the namespace that is spanned by the PIB. Any PIB provides different Provisioning Classes (PRCs) that define the data structures which are used for the corresponding type of policy. This allows for a fine-granular distribution and update process of policy objects but in turn causes a management overhead since the target policy language has to be defined as PRCs and has to be BER encoded. There exist a couple of supplemental Request For Comments (RFC) documents that explain how to define and structure policy data to be used with COPS-PR. RFC 3159 [11] provides the authoritative rules for updating BER encoded PIBs. BER encoding stands for the Basic Encoding Rules [8] for the Abstract Syntax Notation ASN.1 [7]. The Structure of Policy Provisioning Information (SPPI) [11], defines the adapted subset of SNMP's Structure of Management Information (SMI) used to write Policy Information Base (PIB) modules.

In RFC 3318 [6] the authors give a framework for PIBs which defines a set of PRCs and textual conventions that are common to all clients that provision policy using the COPS-PR protocol.

Since it was decided to use XACML as the language to specify the SicAri policies one would have to translate the syntactical structures of XACML into a PIB and thus into ASN.1 syntax. Furthermore this mapping has to be well-defined and elaborated. The advantages that COPS-PR offers does not seem to legitimate such substantial overhead especially because plain COPS in configuration mode suffices the SicAri requirements as well.

8.1 COPS in configuration mode

While the predominant usage of COPS is the outsourcing mode where the PDP server answers policy decision requests from the PEP the COPS protocol explicitly provides the possibility to

requisition a whole configuration for a component. Because COPS is independent of any policy type or language that it transports this component can either be a router interface or a webservice object as in the SicAri case.

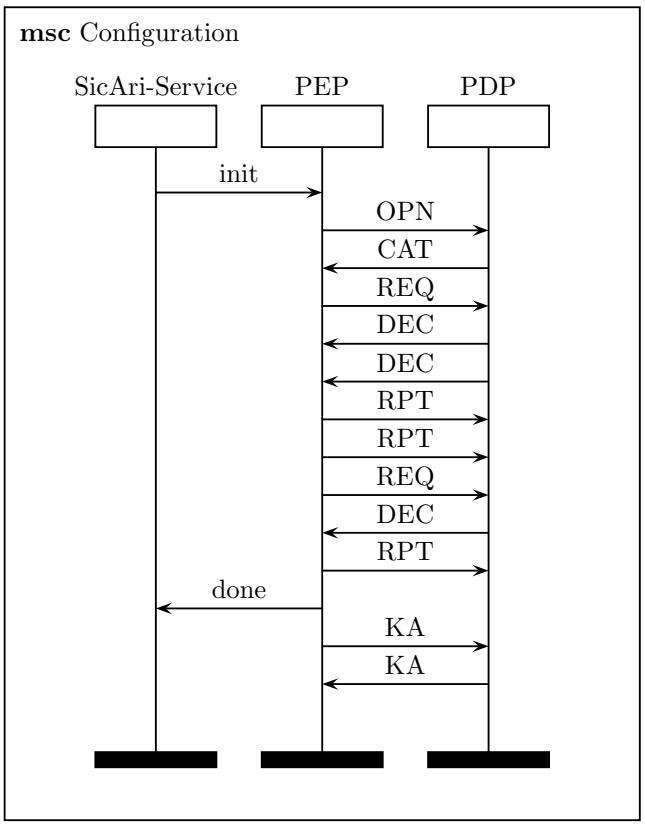

Figure 6: Configuration request

Figure 6 shows the schematic sequence of the COPS configuration procedure. When a SicAri service is started for the first time it contacts the PEP. The PEP sends a client open message (OPN) to the corresponding PDP. This message contains a unique ID that identifies the PEP to the PDP and it also contains a client specific information object (ClientSI). This object helps the PDP to relay the open message to a PDP module that can handle the requests for this particular client type. In case of the SicAri framework this will be an XACML.

When the PDP is capable to serve the client type it answers with a client accept (CAT) message and expects incoming requests. In the configuration scenario the PEP sends a stream of request messages (REQ) that contain context objects which identify the message as configuration requests. The request messages also comprise ClientSI objects that carry client specific information on the requested configuration data. It is up to the client type specification to define a way how the requested configuration data is structured, addressed and identified. One COPS REQ message may order the whole XACML configuration for a SicAri service or it may only requisition separable parts of the XML structure. A supplementary document to this report will discuss these questions and define the XACML client type for COPS.

Each configuration request may be answered with a single decision message or a stream thereof. On reception and successful installation of the configuration data the PEP acknowledges this to the PDP with report state message for each of the DEC messages.

When the PEP finally has received all configuration data from the PDP it signals the installation back to the SicAri service which in turn can use the LPDP to decide its policy requests. The PEP keeps up the connection to the PDP as long as the SicAri service remains active. For that purpose it regularly excanges keep alive (KA) messages with the PDP.

8.2 XACML over COPS

The COPS standard [3] defines a query and response protocol to support policy control for networking. The framework is designed to be extensible and does not make any assumptions about the type of policies to be transported between the COPS clients and servers.

In the SicAri project we extend the COPS framework with an XACML client type. All COPS messages start with a common header that determines the message type and the payload type. Figure 7 shows the schema of this header whose relevant fields are described below.

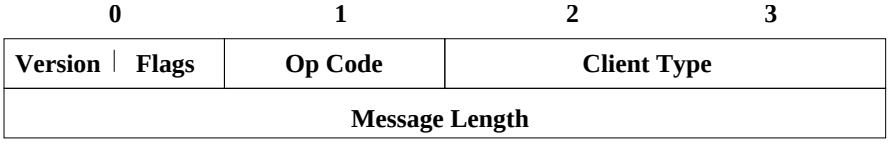

Figure 7: Common COPS header

The Op Code field indicates the type of the message like Request or Keep-Alive. The Client-type field carries a 16Bit number that uniquely identifies the payload that is carried in this message. For example a client-type of 1 defines that COPS transports RSVP policy data ([2]). In the section "IANA Considerations" of [3] three different blocks of numbers are declared. IANA stands for the Internet Assigned Numbers Authority which registers and tracks numbers that have to be uniquely identifiable in the internet universum. These are numbers like port and protocol numbers.

The first group of Client-type numbers covers the range from 0x0001-0x3FFF and its member must be registered with IANA and they require a published COPS extension document. The second group from 0x8000 - 0xFFFF is registered and tracked by IANA but IANA does not assure uniqueness of the numbers nor does it demand published documents for these Client-types. We chose to pick the Client-type number for our XACML extension from the third range that is reserved for private use. These numbers are neither registered with IANA nor are they ever supposed to be used in standards or released in products. With the progress in the SicAri project it may be reconsidered to register a number for the Client-type with IANA or even propose a standardization of it. Nevertheless the COPS usage for XACML will be documented in an extra report that will accompany the SicAri implementation.

Each COPS message may consist out of different COPS objects. Section three of the COPS standard defines the contents of the protocol messages and determines their mandatory and optional parts. The message content is encoded with the help of 16 different predefined COPS objects. Some of these objects provide fields to carry client specific data like error-codes or reason-codes that signal why a particular request has to be deleted. The most important object is

the afore mentioned Client Specific Information Object (ClientSI) that has variable length and transports the client-type data.

9 Policy Generation

This section will be provided in the next version of this document.

10 Outlook

After numerous discussions within the SicAri‘s policy integration team, we came to the conclusion, that for a platform, such as the SicAri platform, which is highly distributed, asynchronous and modular and using security policies in different ways and on different levels of description, the traditional concept of security policies may not be sufficient. Security policies may for example describe

- access rules specifying which entities have (not) access to specific resources,
- security settings for different classes of security,
- security work flows (i.e. consideration of context information within security decisions),
- how to proceed, if the policy engine is not (yet) available (e.g. while the platform is starting up),
- whether policy decisions can be made on the basis of locally cached policies (i.e. specifying the freshness of cached policies),
- how to process in the case that policies cannot be updated (i.e. whether tot use the old policy, or e.g. deny all access).

When putting all these aspects into the same security policy then we are caught in a vicious circle. We cannot specify fallback mechanisms in the security policy for situations in that we cannot access exactly that security policy.

Thus, we are planning to introduce a new concept, the so called *meta policies*. They include rules and obligations — in contrast to security policies — for special or exceptional circumstances such as start-up of the platform, or instantiation or breakdown of the policy engine. Meta policies specify security rules on a higher level of description. They act on top of the traditional security policies. Meta policies may overwrite security policies or specify logical operators and priorities for solving conflicting policy results. The application of meta policies during the platform's operation may additionally be an indicator for the healthiness of the SicAri platform and/or for the the quality of the security policy specified. We will investigate the use of meta policies within the SicAri platform.

References

[1] *Sun's XACML Implementation, Version 1.2*. http://sunxacml.sourceforge.net/.

[2] J. Boyle, R. Cohen, D. Durham, R. Rajan, and A. Sastry. *COPS usage for RSVP*. , United States, 2000.

[3] J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. *The COPS (Common Open Policy Service) Protocol, RFC 2748*. , United States, 2000.

[4] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. *COPS Usage for Policy Provisioning (COPS-PR), RFC 3084*. , United States, 2001.

[5] FlexSecure GmbH. Deliverable AW2.2: Design und Integration der Out-Of-The-Box PKI — Architekturbeschreibung. Technical report, SicAri Consortium, 2004.

[6] S. Hahn, K. Chan, and K. McCloghrie. *Framework Policy Information Base, RFC 3318*. , United States, 2003.

[7] International Organization for Standardization. *Specification of Abstract Syntax Notation One (ASN.1)*, December 1987. International Standard 8824.

[8] International Organization for Standardization. *Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, December 1987. International Standard 8825.

[9] International Telecommunication Union (ITU-T). *X.501 Information technology — Open Systems Interconnection — The Directory: Models*.

[10] International Telecommunication Union (ITU-T). *X.509 Information technology — Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*.

[11] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. *Structure of Policy Provisioning Information (SPPI), RFC 3159*. , United States, 2001.

[12] J. Oetting, J. Peters, U. Pinsdorf, T. Rochaeli, and R. Wolf. Deliverable PF3: Specification of the SicAri Architecture, Version 2.0. Technical report, SicAri Consortium, Oct. 2005.

[13] J. Oetting, T. Rochaeli, and R. Wolf. Deliverable PF2: Requirements for the SicAri Architecture. Technical report, SicAri Consortium, Jun. 2004.

[14] J. Repp, R. Rieke, and B. Steinemann. Evaluierung von sicherheitszielen auf basis von policies. Technical report, SicAri Consortium, Aug. 2005.

[15] T. Rochaeli, R. Rieke, and R. Wolf. Deliverable Pol3: Policy Patterns and Its Application. Technical report, SicAri Consortium, 2005.

[16] T. Rochaeli and R. Wolf. Deliverable Pol1: Requirements on SicAri Security Policies. Technical report, SicAri Consortium, May 2004.

[17] T. Rochaeli and R. Wolf. Deliverable Pol4: Policy Generator. Technical report, SicAri Consortium, 2006.

[18] RSA Laboratories. *PKCS 11: Cryptographic Token Interface Standard*.

A Glossary

ASN.1: Abstract Syntax Notation

BER: Basic Encoding Rules for the Abstract Syntax Notation ASN.1This section will be provided in the next version of this document.

COPS: Common Open Policy Service

COPS-PR: COPS Usage for Policy Provisioning

IETF: Internet Engineering Task Force

LPDP: Local Policy Decision Point

PAP: Policy Administration Point

PBN: Policy Based Network Management

PDP: Policy Decision Point

PEP: Policy Enforcement Point

PIB: Policy Information Base

PIP: Policy Information Point

PRC: Provisioning Class

RAP: Resource Allocation ProtocolThis section will be provided in the next version of this document.

RBAC: Role-based Access Control

RFC: Request For Comments

SAML: Security Assertion Markup Language

SPPI: Structure of Policy Provisioning Information

SSL: Secure Sockets Layer

TCP: Transmission Control Protocol

XACML: Extensible Access Control Markup Language