3

Basic Concepts of SOA Security

This chapter presents the basic notions and concepts of security. We will elaborate on these in later chapters in context of a motivating case study from e-government in Chapter 5 and an extensive case study from healthcare in Chapter 12.

Based on the many meanings of security we elaborate a definition of security appropriate to the context of distributed and decentralized systems in Section 3.1. We move on to define security concerns in context of such systems in Section 3.2. We introduce the key concepts facilitating the expression of these concerns in terms of security "needs" of an asset: either when engineering and managing security-critical systems (as Security Policies in Section 3.3) or when evaluating them in light of the three driving forces defining the state of a system's security: Vulnerabilities, Threats and Security Controls (as Security Requirements in Section 3.4). We close with Section 3.5 introducing Web Services Security Standards as a means to realize SOA Security.

3.1 What Is (SOA) Security?

Common knowledge defines Security as a state of freedom from risk or danger. It can also mean a state free from doubt, anxiety, or fear. Computer security narrows the focus to computing systems. It describes a field of computer science dealing with risk, threats and mechanisms related to the use of computing systems. Even seen in that context, the definition of security comes in (too) many flavours. For example, Garfinkel et al. define computer security in a very broad sense, emphasising the notion of a system's availability [96]:

> *"A computer is secure if you can depend on it and its software to behave as you expect."*

However security obviously does not only describe a desirable state, where systems function as intended. It also encompasses amongst other things - the

notion of actively taking measures to preserve this state through security measures. Gollmann gives a complementary definition [100]. Accordingly, security

> *"... deals with the techniques employed to maintain security within a computer system."*

Nevertheless, these two definitions – even taken together – fall short on one important point. Nowadays, computing systems cannot be viewed anymore as isolated hosts offering computational functionality to human users. Rather, modern computing systems are loosely coupled components distributed over a network and communicating with each other: they are heterogeneous, distributed, and inter-connected. For one, it is evident that a system which is connected to other systems is exposed to a considerable amount of additional security threats. Nevertheless there is another quality in todays computing architectures. Computer systems are not conceived as centralized architectures anymore. A Service Oriented Architecture represents an inherently decentralized computing concept. Hence, an appropriate understanding of the concept of security needs to take into account the system, its context and dependencies between both.

Therefore, the first dimension we need to add to reach a working definition of security for our goals is the dependency of a system on its "surrounding" or context – which in our case refers to distributed and decentralized architectures.

Once we identify all relevant dependencies of a system's well functioning on its context, security can be analysed properly and implemented correctly. These dependencies make security a relative attribute, which may best be understood in terms of how much it contributes to achieve a specific category of "needs". As a consequence, the second dimension we add to our definition of computer security is the relationship to specific goals – in terms of security needs – to be reached.

Accordingly, we define security as (extended and adapted from [100], [53], and [162]):

> *"... the sum of all techniques, methods, procedures and activities employed to maintain an ideal state specified through a set of rules of what is authorized and what is not in a heterogeneous, decentralized, and inter-connected computing system."*

Going beyond the requirements of traditional, monolithical computing systems, this definition of Security addresses the specificities of Services Oriented Architectures in three dimensions:

1. the architecural blueprint of SOA: SOA infrastructures realize security-critical business processes involving many partners and spanning multiple security domains – this significantly augments the number and complexity of security requirements to be met,

2. the organizational aspect of SOA: partners want to stay in control over their part of the workflow, thereby imposing a decentralized, peer-to-peer style architecture, and
3. the administrative challenge of managing and enforcing SOA security: parties may not know each other prior to engaging in a buisness relationship. Some partners may even stay (at least partially) unknown to each other when interacting.

Taken together, these three dimensions clearly justify the need for the definition and analysis of security challenges and concepts in light of SOA. We will take our definition of security as a starting point to clarify relevant concepts of SOA Security in the next Section.

3.2 Security Objectives

Security Objectives provide a categorization of the most basic security needs of an asset. [12] defines a Security Objective as:

> *"... a statement of intent to counter identified threats and/or satisfy identified organisation security policies and assumptions."*

Security Objectives are also called security properties, security aspects, security concerns or security states [172]. Literature categorizes them according to various taxonomies. For example, Bishop identifies three basic aspects of computer security: confidentiality, integrity and availability, whose interpretation vary according to the context in which they arise [53]. Menezes et al. list as many as seventeen basic objectives for information security, among them common objectives such as confidentiality, integrity, identification, and authorization [142]. They also identify signatures, timestamps, and receipts as security objectives, which in conventional terms are rather seen as mechanisms, means to realize a specific objective. However those security objectives are derived by the four cryptographic goals: confidentiality, integrity, authentication and non-repudiation. Eckert identifies six basic security objectives (Authenticity, Confidentiality, Integrity, Availability, Accountability, and Anonymity) [78].

For our purpose, we identify three broad categories of generic Security Objectives according to the basic goal that is pursued for a given asset. We rely on the taxonomy given in [53] and define the three objectives accordingly:

1. Confidentiality is the goal that data should be readable to actors with appropriate permission.
2. Integrity is the goal that data and information should not be altered if not explicitly allowed.
3. Availability is the goal that assets have to be available to authenticated and authorized individuals when needed.

The definition is deliberately kept abstract so to emphasize the independence of any technical, architectural and application level context.

We define other security objectives as mentioned in literature (e.g.,[142]) as *Security Policies* which realize one or more of the three generic Security Objectives. For example Authenticity actually refers to the Integrity of information identifying the sender; a Break-Glass-Policy (BGP) specifying emergency access to sensitive information realizes Confidentiality but also Availability: it aims at making information about who accessed what information availiable (e.g., through logging) in case of a security incident. An introduction into Security Policies is given in the next Section.

In part II, Chapter 7 extensively covers Basic Security Policies realizing exactly one objective whereas Chapter 11 covers Advanced Security Policies realizing a combination of the three basic objectives.

3.3 Security Policies

Security Objectives provide a generic categorization of goals that – when strived for – may contribute to reach a certain kind of security need. Security needs may take either of two forms depending on the context of their use: we subsequently introduce the concept of *Security Policy*, and the next section introduces *Security Requirements*.

A Security Policy realizes a specific Security Objective (or a combination thereof). A Security Policy is defined as [53]:

"... a statement of what is, and, what is not allowed."

We define Security Policies as semi-formal models. Formal in the sense that they can be expressed in a machine-readable way so to configure security mechansims, informal in the sense that we do not provide a mathematical definition. This may give raise to legitimate criticism: policies may not be formulated unambiguously. However, our apporach focuses on applicability and usability in an industrial context. A formal policy may add a considerable degree of precision and even be the only way to prove the policy's correctness at a specific level of assurance. But this usually comes at a price too high with regard to general applicability. We define an effective policy as based on the common consensus and interpretation of the community supposed to use it. Nevertheless, formal models can be integrated in the framework as needed: in later chapters we will integrate models with a proven formal underpinning (e.g., for Role Based Access Control) to support the definition of complex policies.

We differentiate between *Basic Security Policies* and *Advanced Security Policies*. The latter are based on a formal *Policy Model*. We will present the four most prominent Policy Models before introducing Advanced Security Policies.

3.3.1 Basic Security Policies

A *Basic Security Policy* considers one of the Security Objectives (Confidentiality, Integrity, and Availability) in isolation.

Confidentiality Policy

A *Confidentiality Policy* specifies system states where only those entities which are authorized can access information. Such a policy realizes the Security Objective of Confidentiality. Nevertheless it relies on authentication and authorization as a means to realize access control. Whereas authentication is the mechanism to establish and verify an entity's identity, authorization realizes a specific security model on how to grant various privileges to authenticated entities. This is the reason why both – authentication as well as authorization – where not defined as Security Objectives but are mere means to an end. Security Policy Models are covered in Section 3.3.2, Security Mechanisms in Section 3.4.2,

SOA are message based systems. The use of open and machine processable standards makes the messages particular prone to manipulation and unauthorized disclosure. In such systems communication is secured through the use of cryptography. Confidentiality is realized through the encryption at the message-layer. The possibility to encrypt various parts of a message with different keys allows end-to-end security, keeping the message parts confidential and accessible only to the intended recipient even if travelling over intermediaries (or stored with them temporarily). In SOA, a Confidentiality Policy is enforced through standards like XML-Encryption, XML-Digitial Signature and WS-Security (Please, refer to Section 3.5).

Note that we do not interpret such a policy in the sense of guaranteing what is commonly known as an *Information Flow Policy* based on complex mathematical models (e.g., Bell-LaPadula or Biba and Clark-Wilson for Integrity [53]). As we will see in later chapters, we define this policy as preventing the unauthorized disclosure of information in a basic, distributed SOA scenario, where peers exchange documents.

Integrity Policy

An *Integrity Policy* identifies authorized ways in which information may be altered and subjects authorized to alter it. Integrity comes in two flavors: data integrity ensures that data are not compromised and can thus be trusted over a specific period of time, whereas integrity of origin guarantees that information about a recipient is correct. Both are implemented with the same cryptographic primitive: digital signature.

Like message confidentiality, integrity is realized through the application of cryptographic primitives at the message level. In most cases only parts of a message are signed. Besides boosting performance (leaving uncritical parts

unsigned) this also caters to the fact that a message travelling over many intermediary may be subject to many transformations (e.g., adding application-level information during process execution). A message may therefore not pass an integrity check even after a single transformation. This means that integrity must be realized at a level of granularity below the message level (e.g., elements, or parts of a message). An Integrity Policy is enforced through standards like XML-Encryption, XML-Digitial Signature and WS-Security (Please, refer to Section 3.5).

Availability Policy

An *Availability Policy* specifies system states where the provision of a specific resource has to be guaranteed. Availability is not only an important aspect of reliability, guaranteeing the existence of a resource. In security, the aspect of availability is interpreted in the sense of non-repudiation: someone may use a resource, access information, or call a service as needed under specific conditions, and that use must not be deniable.

Non-repudiation is an important security requirement for the realization of SOA executing mission-critical processes. Electronic transactions have to comply with a plethora of legal regulations. In scenarios where partners mostly unknown to each other engage in a business relation, the digital signature is a means to realize a legally binding commitment which holds before court. For example, the Austrian E-Government Law [158] puts the digital signature on a par with its "handwritten" equivalent and specifies requirements for its realization in distributed architectures [91].

In SOA, a *Non-repudiation Policy* is basically implemented through an exchange of signed and time-stamped messages documenting transactions (e.g., the Company sending its annual statement to the Tax Advisor) leveraging standards like XML-Encryption, XML-Digitial Signature and WS-Security (Please, refer to Section 3.5). The variant of our implementation realizing Non-repudiation of Reception as well Non-repudiation of Sending is covered extensively in Chapter 8.

3.3.2 Policy Models

In an industrial context, security concerns usually go way beyond what we cover with the category *Basic Security Policy*. The electronic realization of security-critical processes is tightly coupled with concerns about how to best realize security in compliance with the many provisions, regulations and laws imposed by regional, national, international and industry legislations.

Security Models abstract from specific policies and their particular characteristics. A Security Model represents the formal foundation for an *Advanced Security Policy* with complex characteristics and dependencies between its statements. This model-based abstraction allows a systematic analysis of a policy's correctness and supports systematic reasoning about its properties.

We subsequently analyse four of the most important Policy Models with respect to their ability to cope with complex *Authorization Policies* – which are covered in Chapter 11. Authorization restricts access to authenticated entities holding the privileges to perform an action on a resource. Authorization Policies define the rules of what is allowed and what is not. They are enforced at the various service providers' endpoints through a security infrastructure acting as a single point of entry – a so-called Policy enforcement Point– into the security domains. Prior to granting access to a resource the requester is authenticated and then assigned privileges according to the underlying Security Model. The infrastructure decides by checking upon assigned privileges captured in executable XML policies whether to grant access or not.

In SOA, the administration of these endpoints is a crucial issue. Endpoints not only need to be aware of the technology used to enforce security at the interaction partner's end (e.g., by advertising their technical security requirements through machine-readable policies with WS-Security Policy, cf. to Section 3.5.3), but they also need an efficient concept to dynamically manage these privileges.

The efficient administration of Authorization Policies in distributed environments is a pivotal criteria for the choice of an appropriate Security Model.

Discretionary Access Control (DAC)

In DAC based systems, users in possession of an object are considered to be the owners of the resources. They have full control over the resource. This enables them to use objects they own as they wish, and, for example, to delegate access rights deliberately to any further user [117]. The latter in turn becomes an owner as well and may proceed as he likes.

This notion of *Resource Ownership* makes the DAC model unsuitable for exclusive use in many distributed systems. For example, in a SOA based healthcare scenario, medical data is created by users of healthcare information systems in the domains of various collaborating partners. Evidently, none of those can actually claim ownership of the data. Those to whom the data may be of most value (or, alternatively, those who may suffer the greatest loss when used inappropriately) – the patients – have no control at all.

As a sideline we would like to point out, that the debate about ownership of sensitive medical data is currently gaining momentum in light of technological advances facilitating storage, dissemination, and duplication of sensitive data (e.g., [76], [28] and [139]). The outcome is very likely to be linked to an appropriate definition of the concept of ownership. Ownership can obviously be interpreted in various ways: does the radiologist who actually produced the x-ray own the artefact, the hospital which is bound to store the document for a specific time period, is it your doctor who ordered the x-ray, or is it your insurance who paid for it. The issue is far from resolved and for the time being the intuitive answer may be rooted in the fact that medical data actually is of most value to the individual concerned – the patient – and that he should

be empowered to determine its appropriate use. So in our context, an appropriate security policy model will be evaluated with respect to its abilities to integrate this viewpoint.

The administration of policies based on DAC in distributed environments is evidently almost infeasible.

Mandatory Access Control (MAC)

In MAC based systems, users and their rights are enforced by a central mechanism (e.g., the operating system) and administered by a central authority (e.g., the system adminsitrator). Users do not have the ability to override the policy.

MAC is traditionally associated with multi-level secure systems. The concept of MAC is realized by assigning security labels to data elements at very fine-granular levels, thereby expressing their security sensitivity and assigning clearance levels to subjects. In MLS, less-sensitive information can be accessed by higher-cleared individuals, and higher-cleared individuals can share "sanitized" documents – where sensitive information that the less-cleared individual is not allowed to see is removed– with less-cleared individuals.

MLS was the concept of choice in the mainframe era, where many users had to be granted simultaneous access to sensitive information. It is still in use today in operating systems, however, in many SOA scenarios MAC is of limited use: e.g., inter-organizational healthcare scenarios involve many actors accessing resources scattered over multiple domains and the centralized administration and enforcement architecture of MAC is incompatible with SOA-based Systems which advocate loose coupling with decentralized control.

Role Based Access Control (RBAC)

RBAC enforces access control according to access policies, which define a number of roles and assign permissions to roles [169]. Subjects are assigned one or more roles. A role hierarchy defines inheritance relations between roles. The principal motivation of RBAC – this is to provide administrative convenience – can be further strengthened by using RBAC to manage RBAC ([170]). Many approaches analyse the application of RBAC to workflow management (e.g., [38], [206]) even taking distributed scenarios into account (e.g., [93] and [94]).

The limitations of the basic RBAC model become obvious especially when used in context of Service Oriented Architectures.

Firstly, in practice authorization can generally never be granted exclusively based on permissions assigned to roles as in static RBAC. Rather, access rights depend on a set of dynamic constraints: the right to call an operation of a specific Web service may primarily depend on the caller's role but may be further confined by attributes of the system's environment (e.g., a principal may access a service only between 9.00 a.m. to 5.00 p.m. on working days), of the service call himself (e.g., authentication mode) or on the content of

resources (e.g., a principal with role Tax Advisor may only access the files of Clients he is mandating). Dynamic constraints define the conditions under which a role has the right to access services.

Additonally, the level of granularity necessary for the definition of access rights in security-critical SOA scenarios goes well beyond what is possible with basic RBAC. A requester's permissions may not only be restricted at the services level (e.g., the right of a princpal to call a service that may return a document) but his access rights may have to be further limited at a finer level of granularity (e.g., the permission to only read specific parts of the document). Constraints on permissions support the specification of access rights at various level of granularity.

The framework we are going to present in this book – SECTET – handles these points of criticism by supporting the dynamic constraints through SECTET-PL, a language with predicate logic conditions (cf. Chapter 11).

However, the limitations of RBAC much further: for example, it does not cater for the notion of continuity in access control. This means that it does not support the revocation of access rights once granted. The set of suggestion for improvement is ever groving (e.g., [28]). With these limitations in mind, we will replace the RBAC model with an extended security model – the UCON$_{ABC}$ model in Chapter 12. It is briefly introduced subsequently.

UCON$_{ABC}$

UCON$_{ABC}$ is a comprehensive policy model for usage control. It extends traditional access control models in two respects [160]:

1. continuity of access decision, and
2. mutability of attributes.

Continuity of access decision means that the decision to access an object is not only verified before but also during access and may result in the revocation of permissions, whenever conditions are not met. For example, a Tax Advisor should be allowed to retrieve and read a Client's tax file only as long as he his mandating that client. Once his mandate lost, he should not be able to read the tax file anymore. Policy conditions in UCON consist of subject, object and environment attributes.

Mutability of attributes refers to subject or object attributes changing as side-effects of accessing a resource. This may additionally result in a change in ongoing or subsequent access decisions. This facillitates for example a policy, where access is confined by access history. For example, a Chartered Accountant working for a Tax Advisor should not mandate a Company that stands in direct competition to any of his former clients (a so-called Chinese Wall Policy [53]).

Policy statements in UCON consist of *authorizations*, *obligations* and *conditions*. Authorizations refer to predicates based on subject or object attributes. Obligation actions are directives to a subject to perform additional

actions before or during access. Predicates exclusively based on environment attributes such as system time, device type etc., are categorized as conditions. *A*uthorizations, o*B*ligations and *C*onditions are collectively referred to as the building blocks of UCON$_{ABC}$. UCON conditions can be used to express static constraints (e.g., duration, purpose) as well as dynamic constraints (e.g., number of times to access a resource, location-dependent access).

In a similar manner to RBAC, UCON uses the concept of server-side reference monitoring for access control and trust management. However it also leverages technologies of client-side reference monitoring to enforce usage control and digital rights management. It divides access rights into functional categories like *Viewing* a resource or *Modifying* a data object (e.g., a Physician has privileges to view and update the Patient's medical record).

3.3.3 Advanced Security Policies

Many industry scenarios impose complex security requirements. In this section we introduce scenarios requiring some *Advanced Security Policies*. Some of these use cases already stand as candidates for near future integration into various industry-concerted initiatives (e.g., "Integrating the Healthcare Enterprise (IHE)"-projects [4]) whereas others represent more an educated guess based on discussions with experts on what the industries – especially e-government and e-health – may be needing in a couple of years. All of them are extensively covered in Chapter 11.

Dynamic Access Control Policies

In many security-critical scenarios, permissions to execute services cannot be assigned statically. Instead, they are associated with a set of *Dynamic Constraints*. Such constraints refer to subject, system or object attributes and are evaluated at runtime.

Referring to our running example, a dynamic constraint could state the following: *"A Tax Advisor can modify any tax file records of Clients he is mandating."* It is evident, that this condition has to be checked at runtime as the undelying facts can change over time.

Delegation of Rights Policies (DRP)

DRP allow a user to delegate her rights to other legitimate users of the system in specific situations with defined limitations. For example, in healthcare a patient referral corresponds to such a policy: *"A Primary Care Physician delegates her rights to access an online Patient file to a Radiologist."*

In other scenarios the patient himself could want to grant access to the specialist using the delegated rights of the primary physician [130]. DRP may be further restricted: the rights of the delegatee may depend on additional information such as her legal status, credentials, purpose, duration etc.

Break-Glass Policy (BGP)

BGP is an authorization scheme granting access in case of emergency. An example policy from healthcare could state the following: *"An attending Physician can bypass routine access control restrictions to a Patient's medical records in order to provide timely treatment."*Hence, treatment can occur without any delay due to administrative or technical complexities (e.g., [173, 116]). As an additional safeguard against misuse access could be logged providing evidence in case of abuse.

4-Eyes-Principle

The *4-Eyes-Principle* is a form of *Multiple Authorization*. It requires two users with a common interest to access the system simultaneously. This principle supports monitoring of the data access, e.g., when one user accesses data the other user monitors it (e.g., [182]). In a healthcare scenario, the 4-Eyes-Principle could state that *"The Patient needs to be present when a Physician accesses her records."*

The physician's access is logged during the visit by some trusted *Proxy Service*. Enforcement of the 4-Eyes-Principle is usually performed indirectly and supported with storing the access record into a logging database for future auditing. Logging and auditing capabilities permit the patient to set her privacy preferences based on access history and support the identification of potential abuse.

Usage Control Policies (UCP)

A UCP is an extension of access control because it does not only control data access but also how accessed data may or may not be used or distributed afterwards. In a healthcare scenario a usage requirement could state that *"Access to tax files is allowed for 5 times only and should last for at most 48 hours, after its first access."*

Qualified Signature

In many e-government applications, a system signature is not legally binding. For example, filing your income tax return online with a typical e-government application, the "technical" signature provided by your application or even the security gateway at the organization's domain boundaries is not sufficient when submitting a document. In such a case legal regulations may stipulate that the signatory be a natural person (e.g, the Austrian E-Government Law [158]). This requirement extends the concept of digital (system) signatures to the *Qualified Signature*, requiring the signatory be a natural person.

Privacy Policies

In healthcare, a patient is still to be considered the owner of his records. He retains legal rights over his medical records. Patients concerned about who may read their records could want to define a "Privacy Policy" restricting access to data even if scattered over many repositories.

In public procurement, Anonymity of Bidders is guaranteed by a specific security protocol and in most cases requires a trusted third party. Anonymity of Bidders can be considered a variant of a Privacy Policy.

3.4 Security Analysis

A *Security Policy* is enforced through one or more *Security Controls*. The dependency between policies and controls needs to be constantly evaluated in the light of Security Objectives targeted, underlying assumptions, and looming threats: this is done during *Security Analysis* – an on-going evaluation process.

3.4.1 Security Requirements

In contrast to a *Security Policy* which is used for the management of security (e.g., during runtime) as a set of statements of what is allowed, and what is not, *Security Requirements* focus on the early stages of enigneering, the elicitation phase.

A Security Requirement is a detailed context-dependent explication of a Security Objective. It breaks a Security Objective down into several more detailed descriptions based on the results of *Security Analysis*.

Security Requirements also play a role when defined in context of systems evaluation: security evaluation techniques (e.g., TCSEC [16], ITSEC [11], CC [12]) guarantee that a system may qualify as a trusted system when meeting specific Security Requirements under specific conditions. Standardized, these methodologies provide a measurement of trust based on specific security requirements and evidence of assurance.

We will perform a detailed Security Analysis in the context of use cases for e-government and healtcare systems in Chapters 6 and 12 respectively.

3.4.2 Attacks

Attacks can inflict some kind of loss or harm on computing systems. At its very core computer-based security revolves around three main forces: *Threats*, *Security Controls* and *Vulnerabilities*.

Potential attacks may be the same for SOA as for traditional information systems inside a particular partner node (e.g., malicious software, buffer overflows, trojans, denial of service attacks, cryptoanalysis). They may be based on the same vulnerabilities (e.g., missing user awareness, flawed code and/or

design, wrong administration etc.), and can thus be countered by the same security mechanisms.

Nevertheless, the properties of SOA open the door to a category of specific threats. These threats have to be countered with dedicated technologies catering to the specificities of SOA. As we will see, most of these technologies – the so-called security controls – are based on SOA and Web services standards. We will cover threats and securtity controls specifically in context of SOA subsequently.

Threats and Vulnerabilities

A *Vulnerability* is a flaw in a system's design or its implementation. It is a weakness that might be exploited to cause a system to malfunction, ultimately resulting in some harm or loss. However, a vulnerability may remain undetected and not be exploited at all.

A *Threat* is a specific set of circumstances that bears the potential to cause loss or harm. A threat remains a potential violation of security. It materializes into an attack when a subject (a person or another system) exploits a vulnerability and attacks the system.

Threats can be divided into four broad classes (cf. [53] and [178]):

1. deception which corresponds to the acceptance of false data
2. disclosure resulting in unauthorized access to information;
3. disruption preventing correct system operation;
4. usurpation leads to the unauthorized control of some part of a system.

In the following we give examples attack vectors that may be launched in the context of SOA architectures and specifically leverage technical vulnerabilities, rooted in the implementation of underlying technologies (as in [91]).

XML-specific Attacks. The use of XML for messaging makes the infrastructure particularily prone to attacks targeting those components processing the messages (be they part of the security- ot the services infrastructure). XML-bombs (XML documents with endless recursions), X-Path injections (a technique used to exploit Web services by crafting malicious XPath queries as user-supplied input) and schema poisoning (a modifications of a message's grammatical structure (XML Schema) leading to inconsistencies) may at least render the infrastructure unavailable and at worst compromise the whole system opening access to unauthroized users. Here, a validation service acting as a security proxy or "filter" to any application service can efficiently counter the threat.

Service Scanning. Reconnaissance (aka footprinting) is the activity necessary to a successful operation against a target (e.g., an application, a host in a network, or a service). It refers to information-gathering behavior that aims to profile the target in order to identify efficient attack tactics. It is

evident, that in SOA publicly available information on services (methods, parameters etc.) in their WSDL files could be used for a systematic analysis for weaknesses, for example through automated tests – so-called fuzzying.

Compromised Services. In a distributed scenario service information needs to be retrieved through a service repository. The service repository may hold information on a manipulated, compromised service. This threat can only be countered by authenticating the service provider and checking upon his trustworthyness.

Replay Attacks. One of the most evident attacks on SOA is based on a central property of Web services: their statelessness. As a consequence of that, an attacker could simply intercept a message from an earlier call for a service request and replay it to that service at a later point in time. In case service requests are coupled with costs (e.g., retrieving a Tax File from a Municipality) this has the potential to inflict serious damage.

A system providing a service never remembers which messages already where processed. This necessitates a mechanism to firstly, authenticate a message, e.g., through a digital signature and a timestamp and, secondly, to provide some application level state information. The latter could be implemented through the security infrastructure which keeps status information on messages received and would simply dismiss a replayed message.

For an exhaustive account on technical attacks on SOA architectures, the interested reader may refer to [177].

Most of these attack vectors leverage technical vulnerabilities. However, the Security Analyses in Chapters 6 and 12 will basically cover application layer threats like **Eavesdropping** through compromised communication channels (e.g., the channel between the Tax Advisor Server and and the Municipality Application Server may be compromised) and **Unauthorized Access** due to faulty configuration (e.g., access to the service sendAnnualStatement is not properly configured; services may thus be accessible to companies offering tax sevices which are not anymore actively involved in the scenario).

Security Controls

A *Security Control* is broadly defined as any managerial, operational, and/or technical safeguard put into place to mitigate identified risks. This rather general definition especially applies when performing a Security Analysis (e.g., during requirements engineering).

For our purposes – when designing, implementing or managing a security-critical system – we narrow down the definition to any technical, architectural, or mathematical concept that counters a specific Threat in order to enforce a Security Policy.

In the context of SOA, Web Services Security Standards leverage these techniques, algorithms and mechanisms, and thereby abstract from specific implementational details (e.g., application programming interfaces, management architecture, protocols etc.).

We hence differentiate between *Technical Security Controls* and *Web Services Security Standards*.

Technical Security Controls are generally categorized according to their security function (e.g., Identification and Authentication, Access Control, Audit and Accountability and Systems and Communication and many more [168]). Each one leverages one or more technical, architectural, or mathematical concept (e.g., public- or secret key cryptography for confidentiality through encryption, for integrity, identification and authentication through digital signatures, message protocols for the establishment of trust and accountability, the use of reference monitor for access control etc.). Technical Security Controls accross various layers and tiers (application, operating system, networking, middleware, database etc.) are extensively covered in literature (e.g., [53], [162], [78]).

Henceforth, we will only cover Technical Security Controls in context of their integration into Web Services Security Standards in the next section (Section 3.5) as well as in later chapters when designing and realizing enforcement architectures for various policies in the use cases (Chapters 8 and 11).

The integration into the various Web Services Security Standards and specifications is also extensively covererd in literature, e.g., in [167], [156], and [114]. For a good overview covering these security standards see e.g., [143].

3.5 Web Services Security Standards

3.5.1 Confidentiality, Integrity, and Authenticity

Basic security objectives targeting message security, like Confidentiality, Integrity of data and Integrity of origin (authenticity) are covered by the three basic security standards: *WS-Security*, *XML-Digital Signature*, and *XML-Encryption*.

OASIS proposed an extension of the SOAP message structure to enable the addition of security features to Web services based messaging: *WS-Security* is the basic building block for secure interactions in scenarios on top of Web services technology [37]. The specification describes how to embed security tokens in the header of SOAP messages. These tokens may be used by senders and/or recipients to digitally sign and encrypt the message or parts of it. WS-Security also specifies how to embed these encrypted and signed parts within the SOAP message.

Example: Figure 3.1 shows how the security infrastructure at the Company's side would first encrypt the application relevant data to some cipher value with a symmetric encrytpion scheme (triple-DES-cbc) and then embed that value as an encrypted string according to the standards XML-Encryption and XML-Digital Signature. The symmetric key is encrypted with the recipients public key based on an RSA encryption scheme.

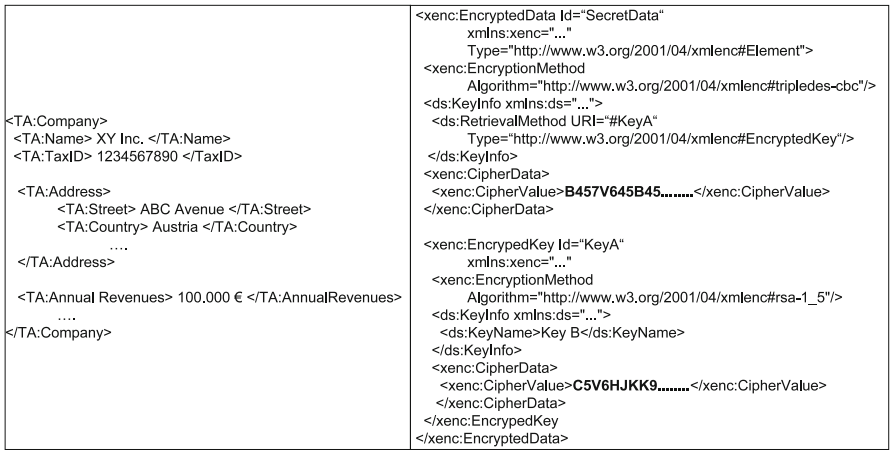

Fig. 3.1. Application Data Encrypted According to XML-Encryption and XML-Digital Signature

In the example, the second cipher value refers to the symmetric key encrypted with the recipients public key based on an RSA encryption scheme. Upon reception and once decrypted, the recipient (Tax Advisor) uses this key to decrypt the first cipher value. This means that the interaction partners are relying on a hybrid crypto scheme to secure their communication. The advantage over using pure public key cryptography lies in better performance during encryption and decryption.

WS-Security in turn relies heavily on the underlying XML-standards *XML-Digital Signature* [43] and *XML-Encryption* [119] for the signing and encryption of XML documents. Both standards specifiy a process for encrypting or signing arbitrary application data and representing the result in XML format.

Example: Figure 3.2 shows how the security infrastructure at the Company's side would then embed the XML structure with the cipher value in the SOAP message structure according to WS-Security before calling the Tax Advisor's service and thereby sending the document.

Vendors and Open Source Initiatives are beginning to offer reference implementation of these standards. For example, *Web Services Security for Java (WSS4J)* [89] is a prototypic extension of the Apache Axis SOAP engine [189] that implements the standard.

3.5.2 Authentication

The process of authentication binds an identity to a subject. Authentication is a technical means to achieve the premises to any non-anonymous interaction.

Authentication is integral to many policies as an implicit prerequisite. For example, an Authorization Policy stating that a Tax Advisor can access his Client's tax files implies the existence of an underlying authentication

```
<?xml version="1.0"?>
<SOAP:Envelope
   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
     <wsse:Security>
      <xenc:ReferenceList>
        <DataReference URI"#bodyID"/>
      <xenc:ReferenceList>
     <wsse:Security>
  <SOAP:Header>
   <SOAP:Body>
   <xenc:EncryptedData Id="bodyID"
         xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
         Type="http://www.w3.org/2001/04/xmlenc#Element">
   <xenc:EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <ds:RetrievalMethod URI="#KeyA"
         Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
     </ds:KeyInfo>
     <xenc:CipherData>
       <xenc:CipherValue>B457V645B45.....</xenc:CipherValue>
     </xenc:CipherData>
   </xenc:EncryptedData>
   </SOAP:Body>
</SOAP:Envelope>
```

Fig. 3.2. Encryted XML Embedded in SOAP Message According to WS-Security

mechanism checking the identity of the requester. Before a request can be evaluated and access granted or denied the requester has to be authenticated. In our context authentication is neither a Security Objective nor a Security Policy. It is considered a technical concept.

Authentication may generally be performed based on:

- something the subject may know (e.g., pin, password, pass phrase, shared secret),
- something the subject may possess (e.g., key, card, token), or
- physical attributes of the subject (e.g., biometrics).

In SOA, authentication occurs at the application- or the SOAP layer instead of relying on transport- and HTTP-layer authentication schemes (e.g., SSL and TLS). The reasons are, for one, that transport layer security is limited to point-to-point interactions, and two, that applications cannot directly retrieve security context information (username, role and/or password of requester) from the transport layer.

Leaving authentication to the application inevitably leads to interoperability problems: in a distributed, heterogenous environment the various applications at the endpoints will, in all likelihood, implement proprietary solutions. Securing communication between all peers turns into a nightmare.

Thus, the most efficient way to realize authentication in SOA, is the integration of security mechanisms in the SOAP message structure. However, SOAP does not provide a specific security model for its protocol. Instead,

it supports security extensions inside the SOAP headers for various security models and mechanisms. WS-Security is designed to incorporate existing security mechanisms for authentication (e.g., X509 certificates, Kerberos tickets, username tokens etc.). The way on how to embed a specific format is defined in token profiles.

Nevertheless, WS-Security does not support much beyond its capability to integrate security tokens. These tokens, incorporating security claims, need to be verified, policies need to be advertised, tokens may need to be mapped from one technology to another etc. This is covered by Advanced Web Services Security Standards building on top of the three basic standards.

3.5.3 Advanced Web Services Security Standards

In this section we briefly introduce advanced Web services standards. The specifications cover application level security concerns that go much beyond what can be covered trough the standards presented in the last section. We confine ourselves to a verbal description without giving code examples as the relevant standards will be covered in-depth as needed in the second part of the book.

The *Extensible Access Control Markup Language (XACML)* is an OASIS standard supporting the specification of authorization policies to access (Web) services [147]. The standard defines a language for the formulation of policies and describes the messages for related queries between components of the security infrastructure. It specifies functionalities needed for the processing of access control policies and defines an abstract data flow model between functional components. The *Role Based Access Control Profile of XACML 2.0* extends the standard for expressing policies that use Role Based Access Control (RBAC) with a scope confined to core and hierarchical RBAC [29].

Example: The Tax Advisor may want to control access to his local services by his employees or external parties through a reference monitor acting as a security proxy to Web services. Every service call would be intercepted, the requester authenticated through some credentials (e.g., message signature), and his rights would be checked against a machine-readable policy stored in XACML format. Once access granted, the service request is forwarded to the application.

XACML is closely related to the *Security Assertion Markup Language (SAML)*, which is the XML-based framework for exchanging security assertions [67]. SAML is integrated into XACML as a profile [30]. It supports the integration of further security-related information – so-called assertions – into the SOAP header.

Example: In a slightly more complex scenario involving many parties, the Tax Advisor may only want to authenticate once (e.g., with an identitiy providing third party), to get an assertion in the form of an Authentication Statement and have this information propagated automatically through tokens

when accessing services of further parties (e.g., Municipality, Notary etc.). These parties can check up with the party that issued the assertion.

The *XML Key Management Specification (XKMS)* describes how developers can integrate access to Public Key Infrastructures in order to secure inter-application communication especially in SOA environments [113]. The specification describes how to use XML- and Web services based interfaces and protocols to third parties providing "expensive" cryptographic services. XKMS consists of two parts - the XML Key Information Service Specification (XKISS) and the XML Key Registration Service Specification (XKRSS). The specifications define specific protocols that can be used for the exchange of messages between an XKMS client and an XKMS server implementation.

Example: The Tax Advisor could rely on XKMS messaging to retrieve certificates from a trusted third party (e.g., certificate authority) to verify the signatures of parties he needs to authenticate.

WS-Trust [33] is based on the security mechanisms of *WS-Security* and defines an extensible model for establishing and maintaining trust relationships across security domains. In SOA, trust is usually realized through the issuance, exchange and validation of security tokens, services offered by a Security Token Service. WS-Trust also defines necessary extensions to the SOAP message structure as well as the protocol bewteen parties relying on and offering the services.

Example: Any peer in the said scenario may need to get some credentials in the form of security tokens from a Security Token Service for authentication with his partners. The format for requesting the tokens is described in WS-Trust. Another example would be the request to a Security Token Service to map from one format (e.g., username token) to another (e.g., Kerberos ticket).

WS-SecureConversation offers features for the establishment of a context for secure communication similar to the concept of HTTPS [32]. Instead of leveraging transport layer security it is based on application-level messaging. It is a protocol that uses a concept based on public keys for the exchange of session keys for message encryption and signature. It thereby provides enhanced efficiency.

WS-Federation realizes the concept of federated security, which allows a set of stakeholders to define a virtual security domain [41]. The specification standardizes the way companies share identities with each other. This is the case whenever authentication and authorization systems are spread across corporate boundaries. Together, *WS-Trust* and *WS-Federation* provide a model to create and broker trust within and across federations.

Example: The Tax Advisor may want to communicate (over Web services interaction) with regular partners in a more efficient way. He could do so by establishing and sharing a security context with e.g., the Municipality. This would also allow him to derive much more performant session keys to secure communication. This increases the overall performance and the security of subsequent exchanges.

http://www.springer.com/978-3-540-79538-4

Security Engineering for Service-Oriented Architectures
Hafner, M.; Breu, R.
2009, XVI, 248 p. 124 illus., Hardcover
ISBN: 978-3-540-79538-4