Security Engineering with Patterns

Markus Schumacher[1] and Utz Roedig[2]

Darmstadt University of Technology

Department of Computer Science

[1] IT Transfer Office (ITO)
markus.schumacher@ito.tu-darmstadt.de

[2] Industrial Process and System Communications (KOM)
utz.roedig@kom.tu-darmstadt.de

27th July 2001

Abstract

Conducting digital business requires secure network and application architectures. The recently increasing occurrence of severe attacks has shown, however, that we will still need quite some time and effort to reach security standards of IT systems alike the standard already usual in other fields. At present, there is a huge gap between theory and the code of practice. Whereas scientists work on formal approaches for the specification and verification of security requirements, practitioners have to meet the users' requirements. The *Pattern Community* recognized this problem, too. Patterns literally capture the experience from experts in a structured way. Thus novices can benefit from know-how and skills of experts. Hence, we propose to apply the pattern approach to the security problem. We show that recent security approaches are not sufficient and describe how *Security Patterns* contribute to the overall process of security engineering. A *Security Pattern System* provides linkage between Security Patterns. Thus dependencies between specific security problems can be considered in a comprehensive way.

1 Introduction

Conducting digital business requires secure network and application architectures. The result of a well-engineered security system must be an architecture that ensures specific security aspects such as privacy, confidentiality, integrity and availability. The occurrence of the *Love Letter* virus or the *Distributed Denial-of-Service* attacks against famous web sites in the beginning of 2000 has shown, however, that we will still need quite some time and effort to reach security standards of IT systems alike the standard already usual in other fields. The following examples show that we obviously do not learn from past security incidents and that we make the same mistakes over and over again:

1. **Virus Attacks.** In March 1999, the *Melissa* macro virus within a Microsoft Word document having been distributed via the email program Microsoft Outlook, caused severe damage (about 80 million dollars). The reason was that Microsoft Word was able to (mis)use the e-mail interface of Microsoft Outlook. In March 2000, the *Love Letter* virus caused a damage of 10 billion dollars. This virus used almost the same methods as the first one, i.e. the e-mail functions of Microsoft Outlook could be used to spread all around the world.

2. **Virtual Private Networks (VPN).** The specification of the Point to Point tunneling Protocol (PPTP) is considered to be secure. During the implementation for the Windows NT platform some basic mistakes have been made though [SM98]. The establishment of an encrypted link requires a shared secret between both VPN endpoints. Unfortunately, the chosen shared secret based on the Windows password scheme, which is known to be breakable and insecure. As a consequence, the overall implementation of PPTP was insecure.

A very important issue is a more comprehensive view of security since security obeys the rule that a chain is as weak as its weakest link. At present, there is a huge gap between theory and the code of practice. Scientists develop formal approaches for the specification and verification of security requirements of IT-systems. The practitioners follow specific methodologies in order to meet the users' requirements such as performance, usability, and reliability. Unfortunately, security is merely treated as add-on. Furthermore, today's software engineering practices don't consider security sufficiently.

The *Pattern Community* is aware of this problem, too. Patterns are a concept to solve recurring design problems in a literary form. As written in [Cop97] "each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves." Patterns capture the experience from experts in a structured way. Thus novices can benefit from know-how and skills of people who have put more effort into understanding contexts, forces, and solutions than novices have done or want to do.

As there is a visible deficit in security engineering, we propose to apply the pattern approach to the security problem. A *Security Pattern System* provides linkage between various *Security Patterns*. Thus dependencies between specific security problems can be considered in a comprehensive way. As shown in this paper *Security Patterns* have several advantages:

- Novices can act as security experts.
- Security experts can identify, name and discuss both problems and solutions more efficiently.
- Problems are solved in a structured way.
- Dependencies of components can be identified and considered appropriately.

The remainder of this paper is organized as follows: in Section 2 we describe basic reasons for the problems in the field of IT-security. In Section 3 we compare some well-known methodological approaches for establishing security. We have a look at the benefits and drawbacks of each concept. In Section 4 the concept of *security patterns* is introduced. We redefine a template for security patterns, present some basic definitions and the related work. In Section 5 we envision the possibilities of security engineering with patterns. In order to underline this we show the benefits of this approach with a real world scenario. Finally, a summary of the contribution of this document and an outlook into the future work is given in Section 6. Appendix A contains the pattern system we used in this document.

2 Security and the Human Factor

The main reason for the existing problems is that security engineering is no technical problem in the first place. Security is always implemented by humans on request of humans. Thus the human factor influences IT-security in significant ways:

- **Security engineering by non security experts.** With the exception of cryptography, the science of IT security is quite new and covers multiple disciplines such as operating systems, computer networks and software engineering. It is very difficult to make IT-systems secure as there are many different components and mechanisms involved. In addition, trust relationships change frequently, which makes an analysis of all security requirements very hard. One cannot assume that the average system developer or architect is a skilled security expert, too. We should notice that in most cases security engineering is done by non-security experts. For example, the developer of an IT-system has to implement the necessary security features to some extent as an add-on. In most cases the developer is an expert regarding the system's functionality, but not regarding its security.
- **Structured problem solution.** When people try to find a solution for a problem, they often follow an ad-hoc approach. The impact on elements aside the adopted solution are overseen in most cases [Dor97]. A good example is the handling of system passwords: a system administrator wants to prevent that the users chose weak passwords. Thus he configures the system to enforce strong passwords, i.e. a user cannot enter passwords that don't meet certain criteria (password must not be contained in a dictionary). This solution is straight forward. Unfortunately, the administrator doesn't realize that nobody can remember strong passwords any longer. As a consequence, users might start to write their passwords on a slip of paper that could be found under their keyboard, or even worse, pinned on their monitor.

- **Scope dependencies and completeness.** IT-security is a very complex area with manifold dependencies. Even if all dependencies are identified, it might be difficult to consider all of them appropriately. A Web server serves as a good example: the administrator can figure out whether the Web server software is secure or not. But probably he cannot decide whether the combination of Web server, operating system and hardware components is secure. The overall system is too complex to be considered as a whole. Often, the developer or administrator of a specific component doesn't even want to worry about security outside his area of responsibility, e.g. the Webmaster only cares about his Web server. Security engineering making sense should be possible without knowledge of the whole security scope and all its dependencies.

- **Time dependencies.** IT-Security is also time-dependent. A system that is considered to be secure today may be insecure tomorrow. Statements about the system's security are only valid with time references. For instance, the strength of cryptographic algorithms typically is founded on difficult mathematical problems. If a genius finds a solution for such a problem, security suddenly disappears.

Security engineering could be done more efficiently if people get support with handling the aspects described above. This support should comprise appropriate methods and tools.

3 Methodologies for Security Engineering

In recent years a number of security methodologies have been developed to assist organizations in establishing and maintaining security. These methods differ regarding their *structure*, *granularity*, *flexibility*, *usability*, *costs*, or whether they are *formal* or *informal*. In this section we outline the characteristics of some well known methods. Then we describe how the aspects stated in the previous Section are addressed.

3.1 Security Policy

Security activities usually begin with the development of a security policy. In general policies describe the use of an institution's information. At a system level, the security policy enumerates objects and assets as well as threats targeted on them. Furthermore, the security-level envisaged is described [CZ95]. Subsequently this security policy is applied to IT-systems. In practice, the security policy is a simple informal text document.

Regarding the problems stated in Section 2, an informal security policy has several serious drawbacks: it does not help to recognize dependencies between various aspects, it contains a monolithic and linear description of problems and solutions, interferences of different sections are not obvious. Furthermore, real-world security policies never reflect the actual security situation and are always out-dated, as many parameters (e.g. requirements or attacks) change frequently. Besides, the person writing the document has to know a lot about security, as no aspect of security should be missed.

3.2 Evaluation Criteria

The evaluation of the security of IT-systems is very important for military, intelligence and more and more public organizations. Thus a variety of evaluation criteria and security guideline documents have been developed by various governments in cooperation with some large organizations. Prominent examples are the *Trusted Computer System Evaluation Criteria* (also known as "Orange Book") and the *Common Criteria*.

All criteria define different levels at which IT-systems can be evaluated and compared. The levels represent different sets of functionality and an increasing level of assurance. The primary goal is to prove that the system fulfills certain requirements regarding its protection mechanisms and that the correctness of the implementation meets a certain assurance level.

However, evaluation criteria have the following drawbacks: they focus on individual *targets of evaluation* (TOE). Therefore dependencies to other IT-systems and components could be missed easily. Besides, the evaluation of IT-systems according to any criteria is a costly and time-consuming task. Thus it is difficult to keep the evaluation results of a TOE up-to-date. Furthermore, it is difficult for the layman to perform such evaluations as the process very complex and requires a lot of background knowledge.

3.3 Tree Representations

Several approaches with symbolic tree representations have been developed. Typically they are often based on *AND/OR* trees that allow some basic calculations if values are assigned to the nodes such as the cost of an attack. The values can be propagated up the tree to the root. Another result could be which attack would be the most likely one, if the values *possible* and *impossible* are assigned to the tree nodes. There are *Fault Trees* (determine system failures that could stop the functionality of a system, see [KM94]), *Threat Trees* (identify threats to a system, see [Amo94]) and *Attack Trees* (identify possible attacks to a system, see [Sch99]).

As the tree approaches cover only single aspects, they can only be a part of an overall security methodology. Tree approaches are a good way to identify threats, faults, or attacks in a systematic way. On the other hand the process of creating any of these trees still requires a lot of security know-how. If you are a novice in the field of security, it is very likely that you will forget some aspects but as written in [Sch00] "creating (attack) trees requires a certain mindset and takes practice" and "you'll get better with time". As a sufficient coverage of all considered aspects (e.g. attacks or threats) is desired, something like a (e.g. public) tree repository is needed. Furthermore, periodical reviews have to defined in order to keep the trees up-to-date.

3.4 Formal Methods

Security is a current hot topic in the formal methods community. Typically formal methods are applied to protocols for authentication, fair exchange, electronic commerce, and electronic auctions.

Policies in a formal way and especially authorization policies are used in the area of management of distributed systems. There exists different languages [DDLS01] to describe policies in a formal way and also mechanisms to check consistency and conflicts [SX01].

Typically formal methods are applied only to specific problem areas such as smart-cards or cryptographic protocols. Larger systems cannot be handled with formal methods due to increased complexity and dependencies. They deliver important contributions to the field of security. But the layman cannot be expected to have the required mathematical background. The verification of a system can be only as good as the assumptions that have been postulated. Thus the biggest challenges of the formal method community will always be to improve the specification capabilities. This could be a difficult task as systems become more and more complex. Other exercises will be to push the integration of formal methods into the overall process of engineering, to obtain tool support, and to cope with partial specifications.

3.5 Semi-formal Approaches

Data-, function- and object-oriented modeling methods are counted among the semi-formal approaches. They are characterized through different graphical elements but also linguistic elements. An example of data modeling techniques is Entity Relationship Modeling (ERM), for object-oriented modeling the Unified Modeling Language (UML). The usage of UML class diagrams offer the possibility to describe a role, his characteristics and relations between roles. Statecharts can be used to describe the behavior of elements or roles. Activity graphs, sequence diagrams, and collaboration diagrams are used to describe the cooperation of the different elements of the whole system.

Condensed, regarding to the aspects from Section 2, semi-formal approaches, especially UML, offer a structured way to describe systems and possibilities to show dependencies between elements. The disadvantage of these methods is that an automated validation isn't possible. However, UML can be made more formal by writing constraints in Object Constraint Language (OCL) as part of the model. The advantages of these methods especially of UML over the formal methods are that they are human-readable, depict and useful for non-modeling experts also.

4 Security Patterns

Patterns are a hot topic in the software community. They describe recurring solutions to common problems in a given context and system of forces [Ale79]. Meanwhile patterns can be found in many problem domains, e.g. Human Computer Interaction (HCI) or Teaching and Learning. In this section we define some basic terms and concepts of security patterns. Furthermore, we present the related work.

4.1 Template for Security Patterns

Patterns are a literary format for capturing insights and experience of expert designers and communicating it to novices. Both the definitions of a *security pattern* and a *security pattern system* that are presented below are based on the descriptions that are given in [BMR+96]. We show what additional aspects are necessary to make a pattern to a security pattern.

Definition 1 (Security Pattern) *A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution.*

The more information a pattern has, the more important structure becomes [Vli96]. Structure leads to uniformity of patterns. Thus people can compare them easily and search for information in a systematic way. Less structure means more informal text, which might be fine for casual reading but unacceptable for comparison and reference purposes. Introducing the key elements of *security patterns* we follow the *Mandatory Elements Present* pattern [MD96]. We make use of the terminology given in the Common Criteria [CC99].

- **Name:** Certainly *security patterns* aren't different to "normal" patterns with regard to their *name*. The *name* of the pattern becomes a part of the vocabulary of the community. It should be easy to remember and refer to. A good name should be evocative and give an image what the pattern might be about.
- **Context (and Related Patterns):** Based on a *scenario* the *context* of the *security pattern* is illustrated. The general conditions under which the *problem* does occur and which *forces* do emerge are described. It is useful to list context setting *security patterns*, too. As some *countermeasures* may introduce other *vulnerabilities*, additional *security patterns* should be considered in the *related patterns* section, too. The same is valid for problems that are solved partly or couldn't be considered within the given *security pattern*. That way a pattern hierarchy will be formed.
- **Problem:** The *Problem* statement defines the problem that will be solved by the security pattern. The major aspects of the problem are elaborated by the viewpoint of the *Forces* to be solved. In the field of security a problem occurs whenever a system component is protected in an insufficient way against abuse.

  Generally spoken we have to deal with generic *Threats* i.e. a potential for the violation of security. A threat is a possible danger that exploit *Vulnerabilities*.

  A typical threat action is an *Attack* that lead to security violations such as disclosure, deception, disruption and usurpation. Thus we propose to use Attack Trees (see section 3.3) in order to identify attacks in a systematic way. Other *Forces* could be trade-offs between security and other aspects such as usability and performance, too.
- **Solution:** This section describes the *Solution* of the *Problem*. Appropriate solutions are determined by the *Context* of the pattern. According to certain *Security Objectives* (that may be written down in *Security Policies*), *Countermeasures* have to be applied in order to reduce the *Risk*. It is useful to warn from pitfalls (how does this pattern become an *Anti-Pattern*) and refer to variants of the pattern.

There are also some optional elements that can be applied if they improve the comprehension of a security pattern. The *Aliases* section lists other names by which this security pattern might be known. Certain diagrams can be used to illustrate *Structure* of and *Interactions* between the participants of a security pattern. Security has impacts on many other requirements such as performance and usability. Thus it could be helpful to enlist the *Consequences* of the application of an security pattern. The benefits and drawbacks of a security pattern can be discussed. In order to illustrate the application of the pattern, concrete *Examples* could be provided [MD96]. Useful are code or configuration samples as well as some sketches. Analogies to real-world scenarios are also suitable such as the *Running Example* of a military base that were used in [YB97]. As we have already denoted, security is always hard to proof. In fact it's much more easier to show that something went wrong. We can use a *Counterexamples* section in order to show how the *Security pattern* can be applied in the wrong way, i.e. it becomes an *Anti-Security pattern*.

Analogous to [BMR+96] we prefer the notion of a *security pattern system*. A security pattern doesn't exist in isolation, there are many interactions with other security patterns. The security pattern system describes the relationships and the linkage between individual security patterns. Thus dependencies between specific security problems can be considered in a comprehensive way. As the field of security is very broad, we cannot speak of a pattern language, that implies the complete coverage of every aspect of the problem domain. Informally a *security pattern system* can be defined as follows:

Definition 2 (Security Pattern System) *A security pattern system is a collection of security patterns, together with guidelines for their implementation, combination and practical use in security engineering.*

A formal model of a pattern system, that is independent of a problem domain, has been introduced in [Bor00]. In [Sch01a] we applied a slightly modified version of this model. It turned out that it is sufficient to refer only to mandatory elements of patterns [MD96]. These elements allows to represent pattern systems as directed acyclic graphs and to perform basic reasoning, e.g. if a cycle occurs there must be something wrong with the process of abstraction. Many pattern practitioners think that formal approaches should not be applied to patterns [BMR+96]. Formal models are, however, especially required in the field of security to proof that certain security requirements are met. In 5.2 we describe what we expect from the usage of a formal model in the context of security patterns. As a security pattern system covers only certain aspects of security, it should meet at least the following requirements [BMR+96]: it should comprise a sufficient base of security patterns; all its security patterns should be described uniformly; the relationships between security patterns have to be exposed; it must support the security engineering process; its own evolution must be supported.

4.2 Related Work

We are aware of the following contributions which focus on security related patterns. An up-to-date list of patterns related to security can be found at [Sch01b]. In order to reflect the evolution of security patterns they are presented in chronological order.

- **Application Security.** In [YB97] a framework to build secure applications is introduced. In order to take advantage of the security of underlying systems, an interface for a *Secure Access Layer* is described. The *Single Access Point* restricts the entries into the application. The *Check Point* allows to make appropriate decisions when dealing with security breaches. In order to achieve access control, *Roles* that grant or deny rights are assigned to groups of users. A *Session* allows to distribute global user information such as the user's identity. Eventually two patterns that deal with *human computer interface* (HCI) aspects are provided. According to their *Role* the user's possibilities are restricted with a *Limited View* of legal options or they are given a *Full View With Errors*, when their privileges are not sufficient to perform certain actions.

  Although the *secure access layer* is introduced the linkage to low-level security services such as cryptography isn't stated explicitly.

- **Cryptographic Software.** Cryptography is the classic area of IT-security. In [BRD98] a set of nine patterns for building cryptographic software components is introduced. The focus is on the traditional aspects of security, i.e. confidentiality, integrity, authentication and non-repudiation. The *Information Secrecy* pattern describes how to keep messages secret from an attacker (confidentiality). The *Message Integrity* pattern shows how to prevent that an attacker modifies or replaces messages without the sharing of cryptographic keys. The *Sender Authentication* pattern illustrates how messages can be authenticated with the usage of cryptographic keys. The *Signature* pattern describes how it can be prevented that communicating parties cannot repudiate a message (non-repudiation). Based on this four generic cryptographic patterns, the remaining ones are derived, such as *Secrecy with Integrity* and *Secrecy with Sender Authentication*. Together they form a *Generic Object-Oriented Cryptographic Architecture* (GOOCA).

- **The Authenticator Pattern.** The work presented in [BDdVF99] describes a single Pattern that "performs authentication of a requesting process before deciding access to distributed objects." With respect to security the authors identified relationships to patterns for authorization as introduced in [FH97] and [NG98]

- **Authorization Patterns.** Several contributions deal with authorization. In [HLF00] the security functions of authentication, access control and data filtering in a distributed environment are combined in a framework. This framework consists of the following patterns: Data Filter [FF99], Bodyguard [NG98], RPC Client [HF99] and Authenticator [BDdVF99]. These patterns and a security model structure build the Object Filter and Access Control Framework.

  In [Fer00] some authorization patterns are discussed, namely Authorization Rules and Role-Based Access Control. The clue is that metadata constraints are used to define authorization. Furthermore, architectural levels (based on the Layer pattern [BMR+96]) are defined whereas each level has its own security mechanism and security enforcement. The combination of a layered architecture that includes a metalevel and patterns are a promising approach for a security pattern system.

Picking up this work, a pattern language for security models is given in [FP01]. There are patterns for established security models: the Authorization pattern (access matrix), Role-Based Access Control pattern and Multilevel Security Pattern (Bell-LaPadula). It is suggested how these abstract patterns can be applied to all levels of the layered architecture.

The work on security patterns has evolved over the past years. Today a mixture of single patterns, frameworks and pattern languages is available. The following topics are, however, left for future work:

- As written in [MM97], a sense of scale is missing from most software design patterns. Especially in the field of security it is very important to have several levels of abstraction and different views on IT-systems. It is left for future work to identify a suitable classification scheme for security patterns. Good starting points are the pattern categories that are introduced in [BMR+96]: architectural patterns, design patterns and idioms. Other promising work is the Layer model described in [Fer00] and the OM-AM framework introduced in [San00]. Such approaches help to "put things in perspective and to emphasize the encompassing nature of security [FP01].
- The application of security patterns at other layers has to be considered. For example the *Check Point* pattern [YB97] does also apply at the *system* and *distribution* layer: firewalls are in fact check points.
- Besides, the pattern approach can be applied to many other security concepts such as cryptographic keys, protocols (e.g. SSL) and algorithms (such as hashes, signatures, and encryption) that could be included in a security pattern system (mining and refactoring of security patterns).
- All security patterns have certain commonalities, or "junction points" where they could be grouped together, to form "the" pattern language (or as we prefer to say pattern system) for the security domain. In future we should think about suitable junction points and concepts for merging security patterns should be developed. The layer approach given in [Fer00] and [FP01] provides a good starting point in this direction.

5 Security Engineering with Patterns

In this Section we explain why security patterns are a suitable approach for security engineering. Then we envision the possibilities of tool supported security engineering with patterns. We introduce our latest activities in this area, too. Then we discuss the benefits of this approach with the aid of a real world scenario in the area of secure networks. The patterns used in this Section can be found in Appendix A. Finally we discuss research topics that aren't covered in a satisfactory way.

5.1 Patterns and IT Security

Security patterns are a suitable approach for security engineering. As shown in the following all important aspects that were given in the problem statement in Section 2 can be covered:

- **Security engineering by non security experts.**

  Security patterns capture the know-how and skills of security experts. Thus Security Patterns enable novices to act as security experts. Experts use the *security patterns* as a common and powerful vocabulary in order to deal with security problems and solutions. The members of a *security pattern community* can identify, name and discuss both problems and solutions more efficiently. The systematic analysis of known security breaches will reveal *bad* practices (anti-patterns). Based on this, refactored solutions can be developed and published as best practice *security patterns*.

- **Structured problem solution.**

  Security patterns prevent ad-hoc problem solutions, as they help to find proven solutions in a systematic and structured way. As security patterns are linked to related security patterns, the impact of certain decisions becomes evident. Side-effects and alternative solutions can be identified easily.

- **Scope dependencies and scope completeness.**

Security patterns introduce several layers of abstraction. This allows to have appropriate views on given problems, e.g. an aggregated view of a system and a finer-grained view on individual components. Furthermore the completeness of solutions is given, as you consider subsequent problems when you follow related security patterns. Security patterns capture experience in certain security problem domains and provide linkage between individual security problems.

- **Time dependencies.**

  System requirements and general conditions change over time. As security patterns are linked, the impact of changes to other security patterns becomes obvious in a very natural way. As security patterns encapsulate problems and solutions, a sound security pattern system should not become useless: changes are only limited to few security patterns. Especially higher levels should not be influenced by low-level changes. If security patterns cannot be applied any longer, they have to be refactored in order to reflect new insights.

5.2 Tool Support

The motivation behind working on a formal model for pattern systems is that formality renders them suitable for reasoning and manipulation by software tools. A supporting set of tools that accompanies pattern systems isn't a genuine objective of the pattern community. However, the usefulness of security patterns during the process of security engineering is related to the kinds of tools that are provided to support aspects such as architectural design, analysis, and evolution. Currently we can think of the following benefits gained by the utilization of security pattern toolkit:

- **Maintenance.** A primary task of a pattern tool is the maintenance of security patterns, i.e. creating, editing, publishing, and reading of patterns. Furthermore the references between security patterns could be managed. A tool could support online pattern writing workshops, too.
- **Classification.** With a suitable classification scheme for security patterns a taxonomy of both problems and solutions can be developed. This requires a formal representation of the statements that are expressed in individual forces (in the problem section of the security pattern) and solutions. Similarly a context hierarchy could be developed.
- **Modeling.** Security patterns could be used for modeling IT-systems and architectures. The security aspects of approved reference models for typical scenarios such as *small business e-commerce* and *business partner access to corporate IP backbone* could be compiled.
- **Reasoning.** With a sufficient set of security patterns, some basic reasoning task can be performed. For example you can ask whether there are other solutions for a given problem. When you add a security pattern, you can check, if it is subsumed by another security pattern, i.e. if it is more specialized or not. It is possible to identify equivalent patterns, i.e. alternatives that differ in minor aspects such as the impact of the solution on performance. Furthermore you can find out, if a security pattern solves problems that you haven't considered yet.

We have already started the development of such a security pattern toolkit (see also [PET01] and [Sch01a]). Similarly as in [vW01] the document format is based on a definition that is specified in XML. It includes text as well as pictures and diagrams (e.g. UML sequence charts). We consider to define at least a XML to HTML transformation. Thus any HTML browser can be used to access the security pattern tool and the underlying reasoning engine. Besides we have investigated for suitable reasoning systems. For the time being we decided to follow up description logic systems. *Description Logics* (DL) form a powerful class of logic-based knowledge representation languages. They are used for expressing structured knowledge and for accessing and reasoning with it in a principled way. The biggest challenge will be to find a suitable interlocking of formal logic statements and the human readable pattern texts.

5.3 Scenario: Network Security Patterns

There are different security approaches to protect a communication channel. If you want to apply any of these methods, you have to integrate them into your IT-environment. A typical security engineering task is to take care that you don't introduce weaknesses that make the overall architecture less secure than before. In this Section we show what went wrong during the Microsoft's implementation of the Point to Point tunneling Protocol (PPTP) [SM98]. Then we describe how the application of security patterns could have prevented this accident.

Typically you use a *Virtual Private Network* (VPN) to establish secure communication over public networks (i.e. the Internet). In order to establish such a *secure channel* one endpoint of the tunnel initiates the connection. An *authentication procedure* has to be passed before both endpoints can communicate securely. Thus *authentication tokens* are needed. As the communication takes place over a public network the information has to be protected against eavesdropping, i.e. a suitable *encryption algorithm* is required. When performance is an important issue, symmetric algorithms are preferred. Such algorithms require the same cryptographic key (a *session key*) on both sides of the channel. The *session key generation* is another important issue, too. Figure 1 shows the relationships between these concepts. There are many more specific security patterns such the *Password* pattern which represents an *authentication token* based on the "something you know" principle. Besides *session keys* require a *secure key distribution* protocol, etc. In Appendix A the security patterns and their relationships to other patterns are described in more detail.

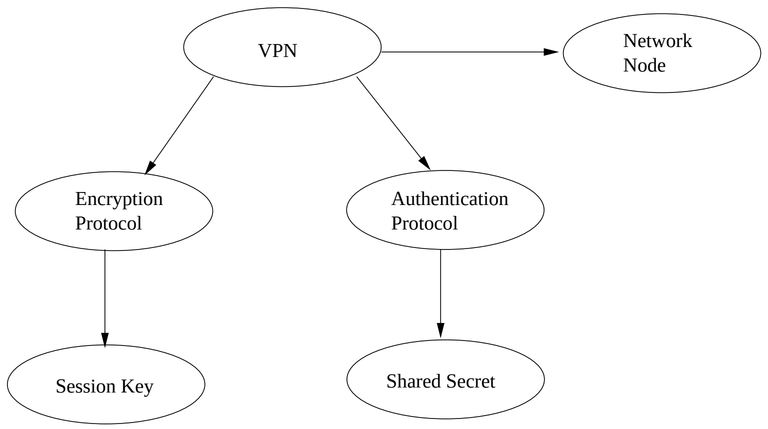

Figure 1: Security Patterns for Virtual Private Network.

The engineers that designed Microsoft's PPTP implemented their VPN solution in the following way:

1. The endpoints of the VPN are *Windows NT* operating systems.
2. The authentication procedure is version 1 of the *Challenge Handshake Authentication Protocol* (CHAP) that uses two password hash values in order to calculate an *authentication token* (login challenge): the *LAN Manager Password Hash* (LANMAN-Hash) and the *Windows NT Password Hash* (NT-Hash).
3. The encryption protocol is the *Microsoft Point-To-Point Encryption Protocol* (MPPE). Here the NT-Hash serves as working basis for the calculation of the session key.

The login challenge can be eavesdropped by an attacker as it is sent without encryption. It turned out that it was very easy for an attacker to extract the LANMAN-Hash. The LAN Manager password is derived from the Windows NT password: all letters are changed to uppercase letters. Thus the hash value can be cracked with brute-force or dictionary attacks easily. As a consequence an attacker knows the Windows NT password, too. Obviously no-one realized that both, the session key and the authentication token are based on the same hash value. An attacker has now access to the client's Windows system and can read the encrypted traffic, i.e. the VPN is broken.

This could have been prevented with a security pattern tool. Analogous to the object-oriented paradigm, patterns can be viewed as classes whereas implementations of patterns correspond to objects. Thus a tool could map PPTP instances to VPN patterns as illustrated in figure 2. The dashed boxes and lines represent instances of security patterns and their relationships. All dependencies and consequences that are described above could be visualized by a tool. The engineer could be notified that the usage of the LANMAN-Hash are a *weak link* that break the chain of security. The impact of the failure could be showed, too. E.g. the tool can show an animation how a security breach propagates through the graph of instances (a cracked LANMAN-Hash affects the client's Windows system and the secure channel). Condensed, the engineer would have seen that the overall PPTP implementation will be insecure. Furthermore, he would have seen that he has to protect the weakest link, i.e. use something better than the LANMAN-Hash and don't use it for both authentication and encryption (in fact exactly this happened - after the flaw was discovered and published by external security experts).

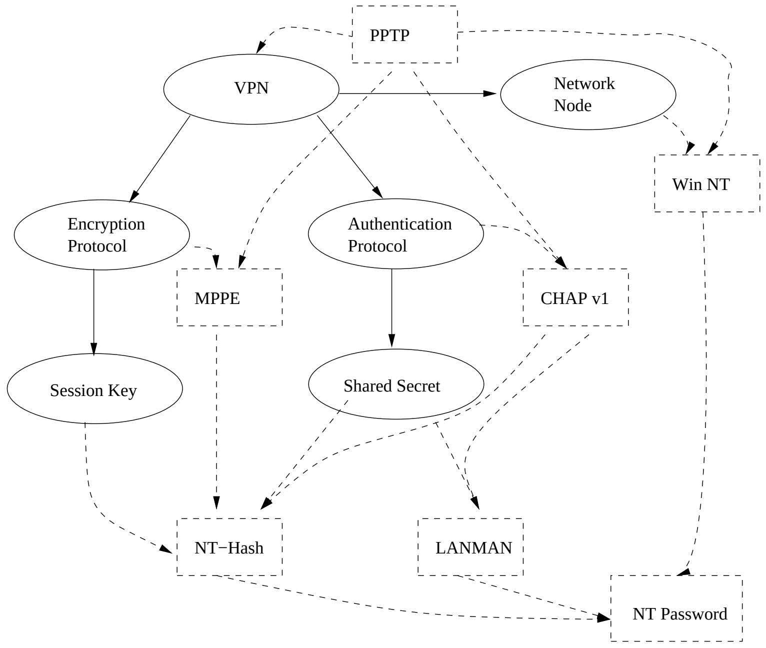

Figure 2: Microsoft PPTP mapped to Security Patterns.

Regarding the aspect in Section 2 this example shows that problems can be solved in a structured way as dependencies between the important components become obvious. The graphical overview helps to identify all problems and to discover weaknesses in an implementation of the pattern system. Even an ordinary engineer who has no experience in the field of IT-security should have been able to get a more secure VPN.

For a true support of security engineering with patterns not all requirements (mentioned in Section 4) are fulfilled in a satisfying way: As we have defined both security patterns and security pattern systems, a uniform description is given. There is not yet a sufficient base of security patterns though. Furthermore there is no definition of a process to support the evolution of a security pattern system. Although the relationships between security patterns are exposed, some research has to be conducted in order to find out more about the meaning of relationships between security patterns (especially with respect of the propagation of weak security links).

6 Conclusions

Security must be a mandatory feature of today's and future IT-systems. Only if we can establish trust in IT environments, we can conduct digital business. Given the current state of the art, the establishment and maintenance of security remains a difficult task. It is necessary to find a way that combines both informal and formal approaches in order to achieve security in a systematic and convenient way. Not only experts, but also average programmers and architects must be able to design secure IT-systems. In this paper we have provided a general discussion of security patterns. We have suggested security patterns in order to cope with the difficulties of the security engineering process:

1. We have identified some important non-technical aspects, that make security engineering difficult. Based on that, we have compared several existing methodologies for security engineering. We figured out that there is a huge gap between security theory and the code of practice. Then we have shown that security patterns are a suitable way to solve today's problems in this area.

2. With a template for security patterns we introduced definitions and characteristics of security patterns and security patterns system. We have presented known security patterns. With a suitable classification scheme they could be merged to a security pattern system.

3. We have shown examples how security patterns can be used. We described what we expect from a set of pattern tools that support the security engineering process. We proposed a couple of network security strategy patterns that should be thoroughly discussed. We showed how they could be merged into a security pattern system.

From our point of view, patterns are a promising approach towards security. A big challenge will be the establishment of a security pattern community that will contribute to the process of security pattern mining, i.e. the identification of security patterns at various levels. On the long run, we want to establish a comprehensive collection of security patterns, as there are only a few activities in this direction today.

We mentioned the benefits of a set of tools that support the security engineering with patterns. A first step will be to establish a security pattern repository that is accessable to the public. On the other hand, tools will (probably) never replace human experts but assist them during the overall security engineering process. As a final remark we want to limit the expectations of security patterns. They are no panacea and don't replace other security methodologies. As an example we proposed to use security patterns in combination with attack trees. Thus security patterns should be treated as a valuable complement in the process of security engineering.

7 Acknowledgements

We would like to thank our shepherd Eduardo B. Fernandez for his valuable comments and suggestions. He helped us to improve our better paper considerably.

References

[Ale79] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979. 4

[Amo94] E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994. 4

[BDdVF99] F. Lee Brown, James DiVietri, Graziella Diaz de Villegas, and Eduardo B. Fernandez. The authenticator pattern. In *PLoP*, 1999. 6

[BMR$^+$96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996. 5, 6, 7

[Bor00] Jan Oliver Borchers. *A Pattern Approach to Interaction Design*. PhD thesis, Darmstadt University of Technology, 2000. 6

[BRD98] Alexandre M. Braga, Cecilia M. F. Rubira, and Ricardo Dahab. Tropyc: A Pattern Language for Cryptographic Software. *PLoP*, 1998. 6

[CC99] CC. Common Criteria for Information Technology Security Evaluation Criteria. ISO/IEC 15408, 1999. Version 2.0. 5

[Cop97] James O. Coplien. A Pattern Definition. `http://hillside.net/patterns/definition.html`, 1997. The Hillside Group. 2

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilly & Associates, Inc., 1995. 3

[DDLS01] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *POLICY 2001*, Lecture Notes on Computer Science, pages 18–38. Springer Verlag, January 2001. 4

[Dor97] Dietrich Dorner. *The Logic of Failure: Recognizing and Avoiding Errors in Complex Situations*. Addison Wesley, 1997. 2

[Fer00] Eduardo B. Fernandez. Metadata and authorization patterns. Technical report, Florida Atlantic University, 2000. `http://www.cse.fau.edu/~ed/MetadataPatterns.pdf`. 6, 7

[FF99] Robert Flanders and Eduardo B. Fernandez. Data filter architecture pattern. In *PLoP*, 1999. 6

[FH97] E. B. Fernandez and J. C. Hawkins. Determining role rights from use cases. In *ACM Workshop on Role-Based Access Control*, pages 121–125, 1997. 6

[FP01] Eduardo B. Fernandez and Rouyi Pan. A pattern language for security models. Technical report, Florida Atlantic University, 2001. Submitted to PLoP 2001. 7

[HF99] Mark Heuser and Eduardo B. Fernandez. Rpc client: A pattern for the client-side implementation of a pipelined request/response protocol. In *PLoP*, 1999. 6

[HLF00] Viviane Hays, Marc Loutrel, and Eduardo B. Fernandez. The object filter and access control framework. In *PLoP*, 2000. 6

[KM94] J. Kohlas and P.A. Monney. Theory of Evidence - a Survey of its Mathematical Foundations, Applications and Computational Analysis. *ZOR- Mathematical Methods of Operations Research*, 39:35–68, 1994. 4

[MD96] Gerard Meszaros and Jim Doble. A Pattern Language for Pattern Writing. `http://hillside.net/patterns/Writing/patterns.html`, 1996. 5, 6

[MM97] Thomas J. Mowbray and Raphael C. Malveau. *CORBA Design Patterns*. John Wiley & Sons, 1997. 7

[NG98] Fernando Das Neves and Alejandra Garrido. *Pattern Languages of Programs III*, chapter Bodyguard (13). Addison- Wesley, 1998. 6

[PET01] PET. The Pattern Editing Toolkit Project. `http://www.ito.tu-darmstadt.de/projects/pet/index.html`, 2001. 8

[Pfl97] Charles P. Pfleeger. *Security in Computing*. Prentice Hall International, 1997. 15

[San00] Ravi Sandhu. Engineering authority and trust in cyberspace: The om-am and rbac way. In *ACM RBAC*, 2000. 7

[Sch99] Bruce Schneier. Attack Trees. *Doctor Dobb's Journal*, pages 21–29, December 1999. 4

[Sch00] Bruce Schneier. *Secrets and Lies - Digital Security in a Networked World*. John Wiley and Sons, 2000. 4

[Sch01a] Markus Schumacher. Merging Security Patterns. EuroPLoP - Focus Group Merging Pattern Languages, 2001. Position paper. 6, 8, 13

[Sch01b] Markus Schumacher. Security Pattern Homepage. `http://www.security-patterns.de`, 2001. 6

[SM98] B. Schneier and D. Mudge. Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP). In *Fifth ACM Conference on Communications and Computer Security*, pages 132–141, March 1998. 1, 9

[Sum97] Rita C. Summers. *Secure Computing - Threats and Safeguards*. McGraw-Hill, 1997. 14, 16

[SX01] G. Stone and G. Xie. Network Policy Languages: A Survey and a New Approach. *IEEE Network*, 15(1), January 2001. 4

[Vli96] John Vlissides. Seven Habits of Successful Pattern Writers. C++ Report, 1996. SIGS Publication. 5

[vW01] Martijn van Welie. The Amsterdam Collection of Patterns in User Interface Design. `http://www.cs.vu.nl/~martijn/patterns/index.html`, April 2001. 8

[YB97] Joseph Yoder and Jeffrey Barcalow. Architectural Patterns for Enabling Application Security. *PLoP*, 1997. 5, 6, 7

A Network Security Patterns

The work on security patterns has evolved over the past years. Today a mixture of single patterns, frameworks and pattern languages is available. In this Section we briefly describe our security patterns used in Section 5.3 and their relationships to the security patterns introduced in Section 4.2.

A.1 Resulting Security Pattern System

In Figure 3 the relationships are illustrated. Each node of the graph represents a security pattern, each arrow indicates a relationship (e.g. a VPN *uses* an Encryption Protocol). Uppercase names indicate that these security patterns are required but have not been elaborated yet. They are left for future work. For convenience we provide pattern thumbnails of these pattern candidates below (Section A.2). The following patterns should be seen as *proposals* and cannot be considered to be complete until now. They should be discussed and developed further by a community of pattern writers. More details about our approach of merging security patterns can be found in [Sch01a].

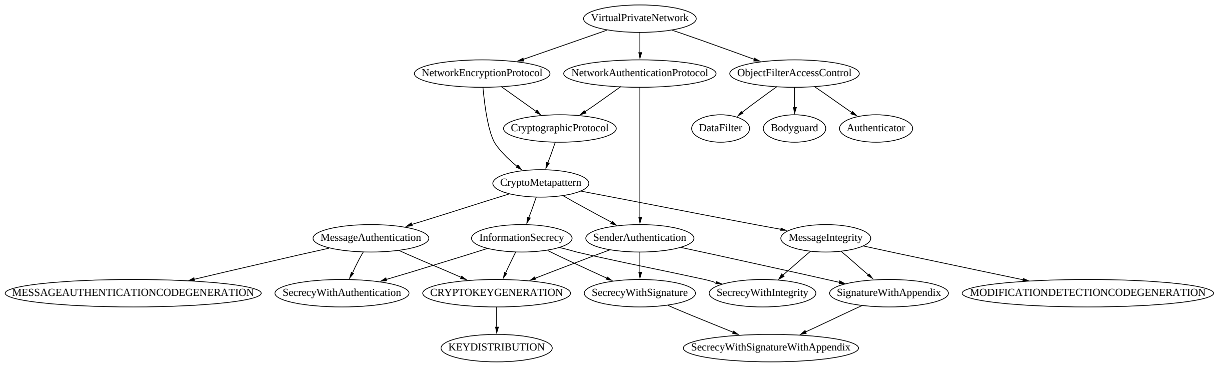

Figure 3: Resulting Network Security Pattern System.

Virtual Private Networks

Context

A number of networks are interconnected via an intermediated network. The intermediated network is only used as a transport medium, thus it is transparent for the attached networks. Appropriate network nodes form the entry and exit points of the intermediated transport network. The interconnected networks combined with the entry/exit nodes form a virtual private network (VPN). The following figure illustrates a generic scenario in this context.

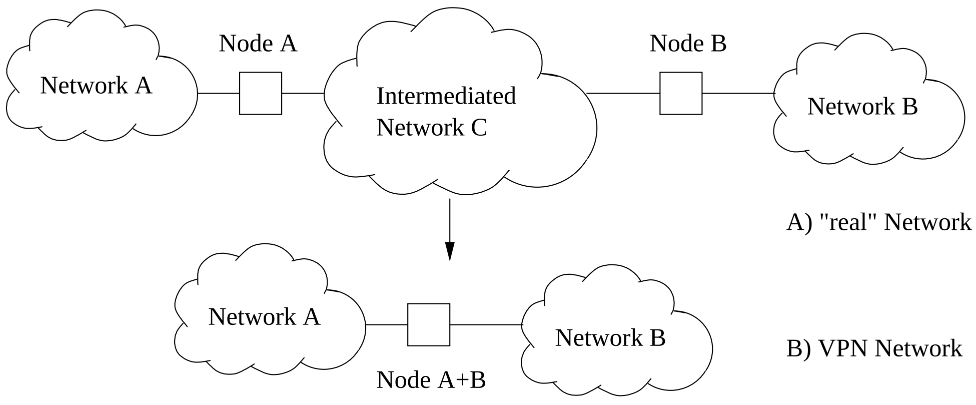

Figure 4: VPN scenario.

Network A and Network B are connected via the the Network C (Figure 4). Node A and Node B are used to interconnect Network A and Network B transparently for hosts in both networks.

Problem

The intermediated network might be not trustworthy such as the Internet. Therefore the transported data between the interconnected VPN parts is exposed to several threats. The forces can be derived from the following attacks to the computer networks [Sum97]:

- **Masquerading of Messages.** Network nodes want to exchange messages *but* how can genuine messages be distinguished from spurious ones?
- **Eavesdropping.** A public network such us the Internet is used *but* how can it be ensured that only authorized users can read messages?
- **Message-stream modification.** You want to exchange messages correctly *but* how can the manipulation of messages be prevented?
- **Repudiation**. You want that your message results in some (re)action *but* how can it be prevented that an attacker repudiates a message?
- **Confidentiality of Traffic Flow.** A public network such us the Internet is used *but* how can it be ensured that an attacker does not even know that a communication takes place?
- **Masquerading of Sender.** Network nodes want to exchange messages *but* how can it be prevented that an attacker masquerades as a VPN network node (e.g. forge source addresses)?

Solution

According to the following security objectives the countermeasures can be derived:

- **Message Authentication.** Prevent masquerading of messages.
- **Sender Authentication.** Achieve non-repudiation of messages.
- **Information Secrecy.** Prevent release of message contents.
- **Message Integrity.** Prevent message-stream modifications.

Use an Encryption Protocol for message protection. Due to performance requirements a symmetric encryption algorithm should be preferred for the data transport. For this encryption algorithm, symmetric session keys are necessary. These keys have to be distributed to the involved network nodes.

- **Confidentiality of traffic flow.** Prevent traffic analysis.

The used mechanisms to ensure confidentiality can also prevent the identification of specific nodes within the attached VPN parts. Nevertheless, the problem of the confidentiality of the traffic flow remains.

- **Sender Authentication.** Prevent masquerading of networks nodes.

Use an Authentication Protocol to establish a trust relationship between network nodes before the exchange of messages.

Related Patterns

A VPN has direct relationships to the following security patterns:

- Encryption Protocols *are used* to achieve protection of confidentiality, authentication, integrity, non-repudiation and confidentiality of traffic flow.
- Authentication Protocols *are used* to establish trust relationships between the VPN endpoints.
- The communicating systems in computer network are called networks nodes. They are the *Authenticator* of users and enforce access control. Thus they can use the *Object Filter and Access Control Framework*.
- The Point to Point Tunneling Protocol (PPTP) *is a* specific VPN implementation.

Cryptographic Protocol

Context

You are transmitting data over a channel that may be intercepted. You apply cryptographic algorithms and mechanism in order to protect message-streams.

Problem

There are several ways to use cryptographic algorithms and mechanism. It must be clear which way the communicating parties use. That way different security features can be achieved with the same underlying cryptographic mechanism (e.g. use public key cryptography for authentication or encryption). The following forces must be resolved:

- **Flaws.** You want to communicate securely *but* an attacker can exploit flaws in the design of the protocol.
- **Behavior.** You want to communicate securely *but* the parties involved in the communication must always know the next step and must be able to detect anomalies.

Solution

A cryptographic protocol is an orderly sequence of steps of one ore more parties to accomplish the protection against threats. A good cryptographic protocol enforces orderly behavior. It has the following characteristics [Pfl97]:

- **Mutually subscription.** All parties agree to the steps of the protocol.
- **Unambiguous.** No one can fail to follow a step due to misunderstandings.
- **Completeness.** A prescribed action is taken for every situation that can occur. If something goes wrong, the protocol should fail securely, i.e. an attacker shouldn't gain an advantage.
- **Established in advance.** The protocol is designed and verified before it is used. If protocols are published early, chances are good that someone will detect a flaw before the protocol is used and severe damage takes place.

Related Patterns

- An Cryptographic Protocol can *use* Information Secrecy, Message Authentication, Message Integrity and Sender Authentication. All these security features are implemented by the Cryptographic Metapattern. Thus we can say that an Cryptographic Protocol *uses* the Cryptographic Metapattern.

Network Encryption Protocol

Context

A number of network nodes are interconnected via an intermediated network. They exchange data over the intermediated network.

Problem

An attacker might be able to access the intermediated network, hence the attacker also has access to the data. Therefore the transported data between the network nodes is exposed to the following threats:

- **Eavesdropping.** A public network such us the Internet is used *but* the transported data can be eavesdropped by an attacker.
- **Masquerading of Sender.** You want to exchange messages *but* an attacker might send data (masquerade itself as sender).

- **Message-stream modification.** You want to exchange messages correctly *but* an attacker might be able to modify transported data.
- **Repudiation.** You want that your message results in some (re)action *but* a sender might falsely deny that it has sent the data.
- **Flaws.** You want to communicate securely *but* an attacker might discover flaws in the protocol.

Solution

According to the following security objectives the countermeasures can be derived:

- **Confidentiality.** The traffic sent through the intermediated network has to be encrypted.
- **Authentication, Integrity and Non-repudiation of messages.** Beside the enforcement of confidentiality, the selected cryptographic mechanisms have also to address authentication, integrity and non-repudiation.
- **Orderly behavior.** The parties involved in the communication must establish conventions how to use the encryption methods. All parties agree to the steps of the protocol. No one can fail to follow a step due to misunderstandings. A prescribed action is taken for every situation that can occur.

Related Patterns

A Network Encryption Protocol has relationships to the following security patterns:

- A Encryption Protocol *is a* Cryptographic Protocol.
- An Encryption Protocol *uses* Information Secrecy, Message Authentication, Message Integrity and Sender Authentication. All these security features are implemented by the Cryptographic Metapattern. Thus we can say that an Encryption Protocol *uses* the Cryptographic Metapattern.

Network Authentication Protocol

Context

A number of network nodes are interconnected via an intermediated network. They exchange data over the intermediated network.

Problem

All network nodes have authenticate themselves to other hosts [Sum97]. We have to address the following forces:

- **Masquerading of Sender** Network nodes want to exchange messages *but* how can it be prevented that an attacker masquerades as a VPN network node (e.g. forge source addresses)?
- **Behavior.** You want to communicate securely *but* the parties involved in the communication must always know the next step and must be able to detect anomalies.

Solution

Mathematical methods can be used to prove identities. The steps of the underlying mathematic algorithm have to be translated into an appropriate network authentication protocol. The parties involved in the communication must establish conventions how to use authentication methods. All parties agree to the steps of the protocol. No one can fail to follow a step due to misunderstandings. A prescribed action is taken for every situation that can occur. After all steps of the protocol are executed a trust relation ship between the involved nodes is established.

Related Patterns

A Network Authentication Protocol has relationships to the following security patterns:

- A Network Authentication Protocol *is a* Cryptographic Protocol.
- An Network Authentication Protocol *uses* Sender Authentication.

A.2 Security Pattern Thumbnails

This Section contains thumbnails of related patterns that are left to future work and are not included in this document. In Figure 3 their names occur in uppercase letters.

- **Cryptographic Key Generation.** A important concept of cryptography is the *cryptographic key*. The key is a shared secret between communicating parties. They are a vital part of cryptographic algorithms that are used to achieve authentication, confidentiality and non-repudiation. There are both symmetric keys and asymmetric keys.
- **Key Distribution Protocol.** Somehow the keys have to be distributed to the communicating parties. This can be achieved by a *key distribution protocol* that is a *cryptographic protocol*.
- **Message Authentication Code Generation.** In order to provide data authentication (and integrity implicitly) message authentication codes are used. Usually a *message authentication* function calculates a hash value of the message with a *cryptographic key*.
- **Modification Detection Code Generation.** In order to provide data integrity modification detection codes are used. Therefore, hash values of the message are calculated.