Security in M3Ci

Norwegian Computing Center
APPLIED RESEARCH AND DEVELOPMENT
Norwegian Computing Center / Applied Research and Development

NOTAT/NOTE

IMEDIA/17/01

Lill-Anita Pettersen

Oslo
August 2001

Norwegian Computing Center
APPLIED RESEARCH AND DEVELOPMENT

Notat / Note

Tittel/Title:
Security in M3Ci

Dato/Date: August
År/Year: 2001
Notat nr/
Note no: IMEDIA/17/01

Forfatter/Authors:
Lill-Anita Pettersen

- **Sammendrag**/Abstract:

The MultiMedia Multi-Channel Infrastructure shall enable the development of multimedia services that can be deployed across multiple devices. The platform functions as a springboard for the prototyping of new products and services relevant to both industrial and public sectors. The infrastructure consists of all generic mechanisms needed by different applications. This makes the development of new applications and services fast and easy. It is also necessary that the infrastructure supports some generic security services required by the applications. Both APIs and underlying protocols have to be implemented.

Emneord/Keywords: security, M3Ci, multi channel infrastructure

Tilgjengelighet/Availability: *Restricted* to channel S Partners and NR until January 2002; *Open* thereafter.

Prosjektnr./Project no.: channel S 11000

Satsningsfelt/Research field: service architecture and service channeling, security

Sensitivitet/Sensitivity: Non-sensitive information

Antall sider/No of pages: 31

Norsk Regnesentral / Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60

Security in M3Ci

Lill-Anita Pettersen

Norsk Regnesentral
August 2001

Table of Contents

Note

Lill-Anita Pettersen

3rd August 2001

Security in M3Ci

Abstract

The MultiMedia Multi-Channel Infrastructure shall enable the development of multimedia services which can be deployed across multiple devices. The platform functions as a springboard for the prototyping of new products and services relevant to both industrial and public sectors. The infrastructure consists of all generic mechanisms needed by different applications. This makes the development of new applications and services fast and easy. It is also necessary that the infrastructure supports some generic security services required by the applications. Both APIs and underlying protocols have to be implemented.

1 Introduction

This note is an overview over different security standards that is possible to use in the M3Ci platform. After a short introduction to the different security services that must be supported, different APIs and protocols will be introduced. Section 7 summarizes what more we need.
Separating security from applications, and store them in the infrastructure, makes it possible to develop new, secure applications quickly.
The note is systemized in two main parts; APIs and Protocols. After this introduction some security requirements for M3Ci are mentioned, followed by a section describing different security services.
After the defintition of security services the first main part starts. The section introduces different security APIs, their benefits and their weaknesses. The second main part consists of a description of some of the security protocols that exist today. The protocol section is divided in four subsections after the OSI architecture: the Network access layer, the Internet layer, the Transport layer and the Application layer. Each layer has it's protocols described. There might be disagreements on which layer a specific protocol belongs, this is one proposition.

2 Requirements

Because of the large scale of applications that the M3Ci platform supports, it is important that all the applications have their requirements satisfied. Since a banking service, that is one of the applications, probably requires the highest degree of security, it is important to have this particular service in mind when deciding for security mechanisms to use in the platform. It would probably be most effective to add the security services that are required by many of the applications in the infrastructure, and services required by few applications in the application itself. This solution depends on the variety of security services required.

3 Security services

In this section different security services which are of interest in the M3Ci platform will be listed. Stallings [15] describes confidentiality, authentication, integrity, nonrepudiation, access control and availabiltiy as a useful classification of security services. This section gives a description of these services and some possible ways to provide them.

3.1 Confidentiality

Confidentiality ensures that a message are only accessible for reading by authorized parties. Confidentiality is achieved by encrypting messages. Messages can be encrypted based on symmetric cryptosystems or asymmetric cryptosystems. In symmetric cryptosystem both parties share the same key, a session key, that they encrypt and decrypt messages with. Only the two communicating parties know this key. Examples of algorithms used for symmetric cryptography are triple DES (the Data Encryption Standard with two or three keys), IDEA (International Data Encryption Standard) and Blowfish. In asymmetric cryptosystems there are two keys: one for encryption and one for decryption. Each party has a private key and a public key. The private key is secret, while the public key may be publicly known. The RSA algorithm and the Diffie-Hellman algorithm is asymmetric. They are mainlyused for key exchange between two communicating parties.
Usually asymmetric encryption is used for key exchange and symmetric encryption is used for encrypting the message.

3.2 Integrity

Integrity ensures that only authorized parties can modify a message. Modification includes writing, changing, deleting, creating etc. One way to provide integrity is by the use of a message digest algorithm (e.g. MD4, MD5).

3.3 Authentication

Authentication ensures that the origin of a message is who it claims to be. The sender is uniquely identified. One way to achieve authentication of a sender is through the use of password, by digital signatures (for example RSA-signatures) or by a authentication protocol like Kerberos as described later in this note.

3.4 Access control

Access control is the ability to limit and control the access to a system. One way to achieve access control is by the use of Access Control Lists (ACLs) which stores the access rights to an object with the object itself..

3.5 Accounting

Accounting is to collect information on resource usage for the purpose of trend analysis, auditing, billing, or cost allocation. One way to store such information is in audit trails. An audit trail is a record of relevant security events.

3.6 Nonrepudiation

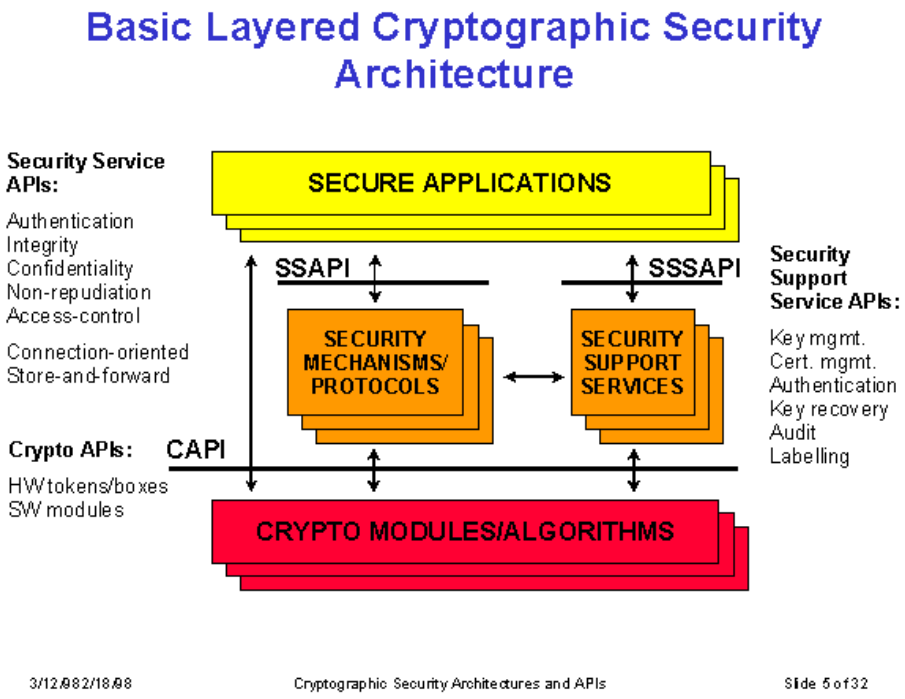
Nonrepudiation requires that neither the sender nor the receiver of a message have the possibilty to deny sending or recieving it. The mechanisms employed to achieve this is encryption, digital signatures and integrity check functions.

4 Application Program Interfaces (APIs)

Petersen and Davie [10] defines Application Program Interface (API):

> Application Programming Interface (API) is the interface that application programs use to access the network subsystem (usually the transport protocol). Usually OS-specific.

They also make a distinction between the protocol that provides a certain set of "services", and the API that provides "syntax" by which those services can be invoked in this special OS. Berkley's unix socket interface is an example on an interface that is widely supported. In this section the APIs discussed are security APIs and not generic APIs as the socket interface. The definition, however, will be the same.
Benefits and weaknesses in the different security APIs will be stated. However, most of the APIs presented in this section are application independent (support future applications as well as todays), module independent (support wide variety of HW and SW modules) and algorithm independent (support



Figur 4.0:Basic layered cryptographic security architecture

future algorithms as well as todays). The APIs give us functional completeness when it comes to security and separate security from application, making it easier to apply new applications. The purpose of this section is to give a description that help choosing the right API for a special application/protocol, since the most suited API to for example a mail application and the best API to a cryptographic token application will differ. One subsection compare four Cryptographic APIs (CAPIs). We will discover that to choose only one API, that covers all the security services we need, is impossible.

Figure 4.0 shows an overview over the layers in a security architecture. The overview tells us that we need more than one layer with security to keep an application secure. In addition to security mechanisms/protocols you need both a higher-level API (represented by SSAPI and SSSAPI in the figure) and a lower-level API, also known as cryptographic APIs (or CAPIs). The API makes it possible for the upper layer applications to determine what services are available at a lower layer.

4.1 Generic Security Service API (GSS-API)

GSS-API provides a simple interface to security services for connection-oriented applications. It provides security services to caller in a generic fashion.
Linn [8] characterises the typical GSS-API caller as a communication protocol calling on GSS-API in order to protect it's communications with security services as authentication, integrity and/or confidentiality. Services available through GSS-API may be implemented over a range of underlying mechanisms. Both symmetric cryptography (e.g Kerberos) and asymmetric cryptography (e.g Simple Public Key Mechanism (SPKM)) may be used.
Mechanism independence is one of four basic goals for GSS-API. Another goal that is addressed is protocol environment independence. GSS-API is independent of the communication protocol used, permitting use in a broad range of protocol environments. GSS-API is also protocol association independent and suitable to a range of implementation placements.

Gollmann [7] defines two logical pieces of GSS-API: the generic interface to the set of security services and a collection of mechanisms providing the security. He also defines the basic security elements in GSS-API which are credentials, tokens, security contexts and status codes. In addition Linn [8] defines mechanism types and naming.
Credentials contain the security-relevant data required to establish a secure context between peers. They have to be handled carefully by underlying mechanisms to maintain security.
Tokens are data elements that are transferred between GSS-API callers. There are two classes of tokens, context-level tokens and per-message tokens. While context-level tokens are exchanged in order to establish and manage a security context between peers, per-message tokens relate to an context and are exchanged to provide protective security services for corresponding data messages. The GSS-API caller who recieves the tokens is responsible for handling them in the right way in order to which type of tokens they represent. This distinction depends on the caller protocol environment.
Security contexts are established between peers using credentials. They capture information related to the management of the security services.
Status Codes are set to support setting up a security context, e.g. a GSS CONTINUE NEEDED status, which indicates that initialising a security context has not been completed. A status flag also indicates which features are desired, e.g mutual req flag to request mutual authentication.
Naming structures are avoided in GSS-API. The names transferred across the interface are treated in order to initiate and accept security contexts as opaque objects.

There exist four types of calls in the interface: Credential management,

Context-level calls, Per-meesage calls and Support calls.
Credential management calls gives the principals the right to acquire and release credentials. Context-level calls manage the security context between peers. Per-meesage calls provides protection of messages. Support calls for routines related to support and general housekeeping e.g GSS Display status which translates status codes to printable form.

GSS-API is more than just a Cryptography Application Program Interface (CAPI). It is a securtiy service API that provides the means to access other security services.

NSA [9] mentions the strengths and the weaknesses of GSS-API. Besides the fact that GSS-APi is algorithm and cryptomodule independent it also supports cryptographically unaware applications. The application programmer need not be an expert in cryptography to implement the API. A weakness when it comes to not supporting aware applications is that it does nor provide a direct interface for services like key management, user authenticated logon, access control and the storage and access to security database information. Since the applications needed to access these services will be cryptographically aware, it will be necessary with another API i addition to GSS-API. GSS-API uses opaque objects to identify sensitive information (e.g credentials, contexts). It is therefore an API that provides safe programming. Some other weaknesses with GSS-API when implemented by itself. It does not support store and forward applications with multiple recievers (IDUP-GSS-API issues this), there is no authentication of the cryptomodule and it does not include auxiliary security services like user logon which makes it more difficult to ensure MISSI support. It is however recommended to use another low-label API together with GSS-API to provide the important things like key-management etc.
GSS-API is a standard.

Implementation: http://www.idoox.com/products/gss/doc/api/index.html

How to use GSS-API: http://devresource.hp.com/STK/impacts/i683.html

4.2 Independent Data Unit Protection GSS-API (IDUP-GSS-API)

Adams [1] describes IDUP-GSS-API as an extension to the Generic Security Service API. In addition to the other facilities provided by GSS-API, IDUP-GSS-API supports store-and-forward and storage applications. IDUP-GSS-API gives the possibility to protect each Independent Data Unit (IDU). The IDU may be of any size and it's protection is entirely independent of any other unit of data.

Adams [1] describes the paradigm within which IDUP-GSS-API operates in as:

> An IDUP-GSS-API caller is any application which works with IDUs, calling on IDUP-GSS-API in order to protect its IDUs with services such as data origin authenticationwith integrity (DOA), confidentiality with integrity (CONF), and/or support for non-repudiation (e.g., evidence generation, where "evidence" is information that either by itself or when used in conjunction with other information is used to establish proof about an event or action.

IDUP-GSS-API is, as GSS-API, mechanism independent, protocol environment independent, protocol association independence and suitable for a range of implementation placements. The basic elements in IDUP-GSS-API are, as in GSS-API, credentials, tokens, security environment, mechanism types and naming. Credentials are the same as in GSS-API. Tokens in IDUP-GSS-API are much the same as in GSS-API, but there are no context-level tokens generated. The security environment in IDUP-GSS-API is different from the one in GSS-API. In IDUP-GSS-API the security environment exists within a calling application. This is because it's purpose is to protect the IDUs using particular caller credentials. Mechanism types and naming are to be understood as in GSS-API.

`Implementation:http://www.meehan.cs.wwu.edu/nw3courses/cs417f/common/gssapi/idup_gss.htm`

4.3 Generic Cryptographic Service API (GCS-API)

Rogaway [11] describes what GCS-API provides:

> It is the purpose of Generic Cryptographic Service Application Program Interface (GCS-API) to provide a simple and abstract interface for gaining access to virtually any cryptographic transformation

GCS-API provides a generic interface to cryptographic operations. Applications are given the possibility to take advantage of cryptography and in that way secure themselves. GCS-API is neither a low-level nor a high-level API. It has the characteristics of both and are therefore called a middle-level API. It can support the underlying mechanism for a high-level CAPI such as GSS-API.

NSAs CAPI team [9] summarizes benefits and weaknesses of the GCS-API.

GCS-API is algorithm independent and application independent. It supports both session-oriented and store-and-forward applications. The cryptographic services provided by GCS-API are authentication, data integrity and confidentiality. GCS-API is the only recommended CAPI for key management applications. It is also recommended for security association protocols and as certification manager.

Some of the basic concepts of GCS-API are mentioned by Rogaway [11]. He describes transforms, cryptographic contexts, keys, quality of service, defaults and security awareness, and naming. Transforms are maps from an integer code, "old state", and zero or more (input) strings, to a "new" state and zero or more (output) strings. The state can be changed or left in the same way as before. Statefull transforms are often used to store keys. A cryptographic context has to be created to use a transform. The creation is done by naming the desired transform and filling it's initial state. A cryptographic context is an instance of keyed transform. A key is an abstract enity to the caller. One can create a key by calling GCS_Init_key, or by using GCS_Retrieve_key. Quality of service for a mechanism is defined by a given transform's transform-type and a set of (attribute, value)-pairs for the mechanism. When it comes to the security awareness there are to possibilities for a GCS caller; the caller can be classified as "security unaware" (U) or as "security aware" (A). Naming structures are, just as with GSS-API, avoided.

More information: http://www.opengroup.org/publications/catalog/p442.htm

4.4 Microsoft CryptoAPI

NSA [9] compares different CAPIs, one of them is Microsoft's CryptoAPI. CryptoAPI supports cryptography in order to secure cryptographical aware applications. CryptoAPI includes functionality for encoding to and decoding from ASN.1, hashing, encrypting and decrypting data, for authentication using digital certificates, and for managing certificates in certificate stores. Encryption and decryption are provided, both using session keys and with public/private key pairs. CryptoAPI is a low-level CAPI developed by Microsoft. The CAPI is not a standard. Hardware and software implementations of the library are called "cryptographic service providers" (CSP). The CSP does the actual interpretation of objects and structures. It also utilizes both hardware and software cryptographic modules. CSPs have to be signed by Microsoft. The kernel contains a 1024 RSA public key that it uses to check the signature when the user tries to load a CSP. If the test fails the user can not load the CSP. One CSP provides a limited set of algorithms and data exchange formats. If the user needs a richer set it have to contact more CSPs to connect to. The CAPI uses the concepts of CSP type to refer to a set of algorithms and associated modes and data formats.

Concepts of a CSP type defines:

- Key exchange algorithms
- Digital signature algorithms
- Key exchange format
- Digital signature format
- Session key derivation scheme
- Key length
- Default modes

CryptoAPI has a default CSP that implements several commercial cryptographic algorithms including RSA and MD5.
Microsoft CryptoAPI is algorithm and application independent. It provides a generic interface to cryptographical services as encrypt and decrypt. It also allows for current and future cryptomodules to be implemented. Many of cryptoAPI's weaknesses are related to it being a low-level CAPI. It is for example weak in the key and certificate area, and should therefore be mixed with an API from a higher level. Since the CAPI is embedded in the OS, it cannot easily be changed by the application developer. Programmers using the API need substantial C-expertise and cryptographic programming expertise. These requirements limit the use of the API where most of the applications are cryptographical unaware and where the application programmer has no C-and cryptographical expertise.

Implementation: http://msdn.microsoft.com/downloads, under security

4.5 BSAFE - PKCS (Cryptoki)

Cryptoki is the fourth CAPI in the comparison done by NSA. BSAFE - PKCS (Cryptoki) is a standard, simple, low-level object oriented CAPI. It is a cryptographic token interface where hardware and software implementations are called "cryptographic tokens". As the only CAPI that interfaces directly to cryptographic tokens, Cryptoki is the logical place for functions that allow user authentication and administrative control over the token. Together with MS cryptoAPI, Cryptoki is recommended for certification manager, security association protocol, ESP, AH, NLSP, TLSP, MSP, S-HTTP, SSL and GULS
Cryptoki is application independent and also, as the three other CAPI's represented in this note, algorithm independent. When developing cryptoki

RSA had personal cryptographic tokens in mind (e.g., smart cards, PC-cards), but there are defined extension mechanisms to allow addition of new capabilities. Cryptoki supports hash, signature and encryption operations. Services as integrity, authentication and confidentiality are provided by these operations. In addition, signature provide support for nonrepudiation.

`Implementation:http://www.phaos.com/e_security/dl_cryptoki.html`
`http://www.phaos.com/e_security/dl_crypt.html`

4.6 CAPI comparison

The NSA Cross Organization CAPI team [9] has done a comparison of four CAPIs: GSS-API, GCS-API, Microsoft cryptoAPI and RSA CRYPTOKI. In this section the team's foundings will be represented. Since the article was related to cryptographic APIs there are some interfaces which are not mentioned, and some characteristics not considered. However, it gives us an overview over four APIs and what the differences between them are.
GSS-API has evolved lately, the extension to the API, IDUP-GSS-API, is the preffered version and will in this section represent the API. As mentioned above the IDUP-GSS-API is a high-level API and will therefore probably be the best choice for application developers. GCS-API, CryptoAPI and Cryptoki should be used only when developing cryptographically aware applications, or as an underlying mechanism to IDUP-GSS-API. IDUP-GSS-API is the recommended API for applications/protocols like word processors, mail applications, file storage, directory seivices, network management and authentication applications.

GCS-API is the proposed API for key management applications/protocols. The GCS-API is special because it has some of the characteristics of high-level APIs (e.g GSS-API) and som from the low-level APIs (e.g Cryptoki). As a middle-level API it has not gain the same suport from the market as both low- and high-level APIs have.

Microsoft CryptoAPI and RSA Cryptoki are two similar APIs. They are both low-level APIs, application independent and require cryptographic awareness from the application developer. CryptoAPI and Cryptoki are both recommended for Security Association (SA) protocols, cerificate managers, ESP, AH, NLSP, TLSP, MSP, S-HTTP, SSL, GULS. Cryptoki is also the recommended API for cryptographic token applications because of it's abstract token model. Table 4.6 shows which API that suits which application.

Which API to choose depends on what protocol/application you have to

Criteria	GSS/IDUP	GCS-API	CryptoAPI	Cryptoki
Algorithm Independence	yes	yes	yes	yes
Application Independence	yes[1]	yes[2]	yes[2]	yes[2]
Cryptomodule Independence	yes	yes	yes	yes
Detailed Cryptographic Awerness Required	no	yes[3]	yes[3]	yes[3]
Design and auxiliary Services				
Key life cycle management	no	yes	no	no
Cryptomodule verification	no	no	yes	no
User authentication	no	no	no	no
Certificate management	some	no	no	no
Query Capability	no	no	yes	yes
Set-up/Tear-down capability	yes	yes	yes	yes
MISSI Suport	yes	yes	yes	yes
Safe Programming	4	2	2	2
Security Perimeter	yes	yes[5]	yes[6]	yes

[1] GSS and IDUP each handle the different application paradigms .
[2] GCS-API, cryptoAPI and Cryptoki are suffiently low-level to be independent of the application
[3] GCS-API, cryptoAPI and Cryptoki are intendend for the cryptographically aware programmer
[4] Safe programming is weighted from 1 through 5, with 5 being the most safe
[5] Keys are protected physically (using hardware) or cryptographically (encrypted under facility master key). Similar protection is provided for intermediate function results (if not kept within the security perimeter) and "exported" contexts.
[6] Microsoft CryptoAPI specifies a programming inteface that supports this function. The degree of support is CSP specific

Table 4.6: CAPI comparison

support. If there are many different applications the perfect API will be difficult to find. The solution then is to mix the existed APIs. Developing a new one is to much work. The recommodation states that in reality, a combination of all four CAPIs will more likely be used. Figure 4.6 shows an overview over CAPIs and which applications thay are suited for. Recommended layered CAPI approach:
-high-level CAPI (e.g., IDUP/GSS-API) for application developers -low-level CAPI (e.g., CryptoAPI, CDSA, or Cryptoki) for cryptographic service developers directly interfacing to HW/SW cryptographic modules.

4.7 Generic Authorization and Access control API (GAA-API)

GAA-API adds authorization and access control to applications in a generic fashion. Ryotov et al [13] write that the API works independently of underlying security protocols and can be used of multiple applications. GAA-API supports the following security mechanisms:

- Security mechanisms based on symmetric or asymmetric cryptosystems

- Different authorization models
- Heterogeneous security policies
- Various access rights

The GSS-API can be used by the GAA-API to obtain principal's identity. While GAA-API provides authorization, GSS-API provides authentication. It will often be necessary to perform both authentication and authorization. Without authenticating an user it is impossible to know that he really is who he claims to be and therefore impossible to decide which right he has. Ryotov and Neuman [12] say that GAA-API is built into applications through a library. The applications use it to decide whether a subject is permitted to do what it wants to do. A simple GAA application will perform some initialization to create a GAA control structure and security context (gaa-get-object-polcy-info). The control structure contains information about callback routines while the security context contains information about the user's credentials. After this initialization the application probably will receive a request. The application will then determine what rights are necessary to fulfill that request and call GAA-API routines to create a list of requested rights (gaa-request-right). When the right policy is found, the application will determine whether or not policy grants those rights (gaa-check-authorization). After the application has finished using GAA-API it calls cleanup routines and release resources. A GAA application's performance is illustrated in figure 4.7. The figure shows an Initiator who wants access to a Target. The Access Control Enforcement Function (AEF), that stands in the middle of the initiator and the target, sends a decision request to the Access Control Decision Function (ADF) with information about the initiator and the target. The ADF decides whether or not the Initiator should get access to the target, and sends the answer to the AEF.

Implementation: http://www.isi.edu/ hochung/research/gaa-api.html

4.8 Java-API

Amit [2] gives a overview of Java security and security APIs. Amit informs that the cryptographic Java API that supports encryption and other code signing tools was first introduced in JDK 1.1x when the new Java Cryptographic Architecture (JCA) became introduced. The new API introduced support for symmetric cryptography, asymmetric cryptography, certificates (based on X.509v3 standard) and one way hashing functions. JDK 2.0 introduced a new security API. The API consisted of two main groups of classes: the access control classes and the cryptographic classes. In addition

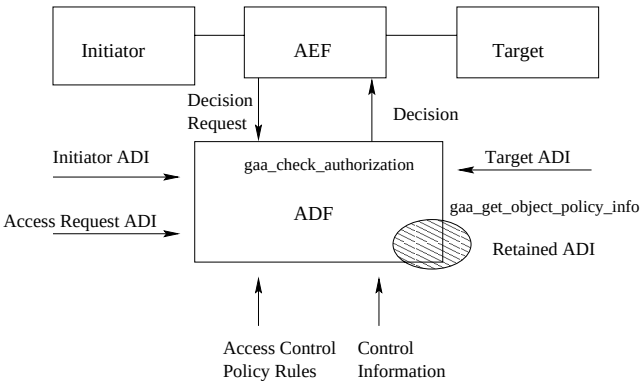


Figure 4.7: GAA API

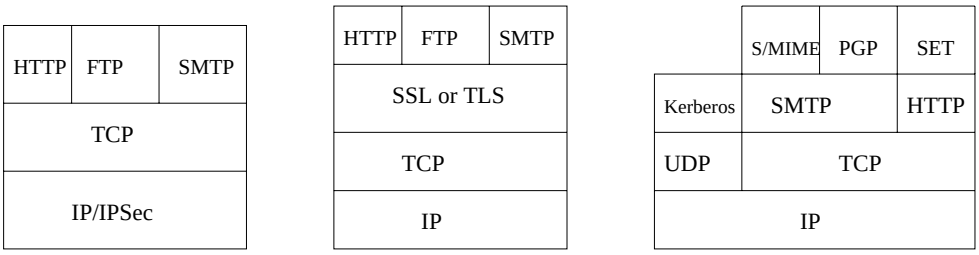
to the cryptographic API version there exist two other JAVA APIs: Java Authentication and Authorization Service (JAAS), that provides user authentication and access control capabilities, and Java Secure Socket Extension (JSSE). JSSE is a set of Java packages that enable secure Internet communication. It implements the Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols, and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. JSSE enables secure passage of data between a client and a server running any application protocol (http, Telnet, NNTP, FTP etc.) over TCP/IP.

Implementations are under the security APIs links.
More information:
http://tetworks.opengroup.org/javadoc/
http://java.sun.com/products/jdk/1.1/docs/guide/security/CryptoSpec.htm
http://www.oop-reserch.com/download.html
http://java.sun.com/j2se/1.4/docs/guide/security/index.html
Java-APImap:http://www.onjava.com/pub/a/onjava/api_map/

4.9 Secure Distributed Environment (SecuDE)

SecuDE is a collection of APIs working side by side.
The following APIs are part of SecuDE:

- AF (Authentication Framework and Certification), adds the functions of a X.509 certificate to the collection of APIs
- CRYPT (Cryptographic algorithms)
- GSS (Generic Securitys Serivices)
- PKCS (Public Key Cryptography Standard)



(a) Network level (b) Transport Level (c) Application level

Figure 5.0: Location of security protocols

- PEM (Private Enhanced Mail Support)
- SECURE (Personal Security Environment and Cryptography
- SEC-SC (SmartCard Plug-in API)
- SCT
- S/MIME (Secure MIME)
- BAKO

Some of these APIs are further described in this note, but not all.

Implementation: http://www.darmstadt.gmd.de/secude/navreg.htm

More information: http://www.darmstadt.gmd.de/secude/Doc/

5 Protocols

This section is systemized in four subsections: Network Access-level Protocols, Network-level Protocols, Transport-level protocols and Application-level protocols. The security protocols located at different levels have different advantages and disadvantages. There are also a number of protocols at each level, and therefore even more difficult to choose the right ones. Figure 5.0 shows some protocols and their location in the stack. The protocols will be further introduced later in this section.

5.1 Network Access Layer

5.1.1 Point-to-Point Protocol (PPP) Authentication

Cisco [4] gives an introduction to authentication using PPP. A PPP session starts with establishing a link between two nodes. If the establishment suc-

ceeds, an optional authentication takes place. PPP supports two authentication protocols: Challenge Handshake Authentication Protocol (CHAP) and Password Authentication Protocol (PAP). PAP works as a two-way handshake where the username and password is sent in clear text. CHAP on the other hand uses a thrre-way handshake. After a PPP link is established , the host sends a "challenge" message to the remote node. The remote node calculates a value using a one-way hashfunction and responds. If this value matches the value calculated by the host, the authentication is acknowledged. PPP CHAP is a standardized protocol for authentication.

`Implementation:http://www.cisco.com/warp/public/131/ppp_callin_hostname.html`

5.1.2 Layer 2 Tunneling protocol (L2TP)

L2TP provides secure tunnels at the transport layer. The protocol is an extension to the PPP protocol that enables ISPs to operate Virtual Private Networks [ms]. L2TP consists of two pieces: the L2tp Access Concentrator (LAC) and the L2TP Network Server (LNS). LAC physically terminates a call and LNS probably authenticates the PPP stream [linux home site]. L2TP consists of the best features of PPTP (from Microsoft) and L2F (from Cisco Systems). The protocol requires that ISPs support it.
home site: http://www.marko.net/l2tp/

5.2 Internet Layer

Protocols at the Internet layer have the benefit of each packet being encrypted individually. Protocols implemented as low in the OSI architecture as the Internet layer provide a high degree of security.

5.2.1 Internet Protocol Security (IPsec)

IPSec is IETF's proposed standard for real-time communication security. The protocol provides authentication and encryption at the IP layer and it's goal is to make the essence of Internet secure. The protocol consists of three important elements; the Encapsulated Security Payload header (ESP), the Authentication header (AH) and the Internet Key Exchange (IKE). ESP supports both encrypted transmission and authentication. AH provides authentication, and IKE manages key exchange.
Stallings [15] says that IPsec supports a variety of applications since it encrypt and/or authenticate all traffic at the IP level. Therefore all distributed applications, including remote logon, client/server, e-mail, file tranfer,web access and so on can be secured. One of the greatest benefits of IPsec is it's placement in the ISO protocol stack. IPSec may be seen as a protocol

between the transport layer (TCP/UDP), and the network layer (IP). Because it is below the transport layer it is transparent to applications. It is therefore possible to include IPSec in the kernel without changing the applications.

The services IPSec provides are:

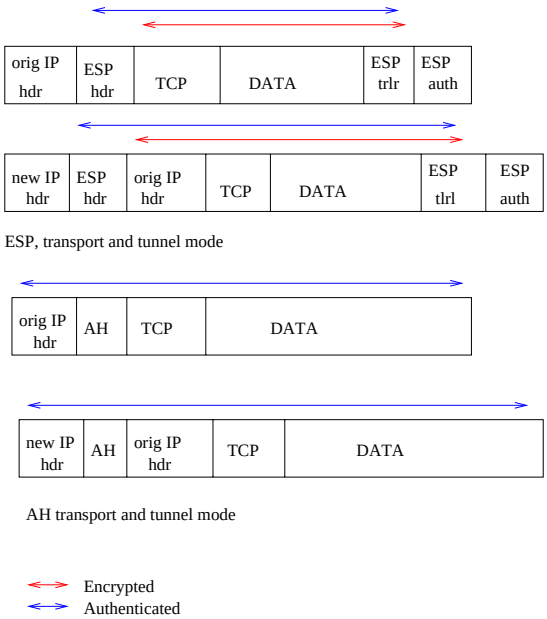
- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets
- Confidentiality
- Limited traffic flow confidentiality

Security Associations (SA) is a one-way relationship between a sender and a receiver that affords security services to the traffic carried on it. Security Associations are important in both AH and ESP. Each SA is unique and affords different security services (ESP or/and AH)
ESP and AH support two modes of use: transport and tunneling. Transport mode is typically end-to-end communication between to hosts. It primarily protects upper-layer protocols. Tunnell mode provides protection of the entire IP-packet. The IP-packet is treated as an inner packet to another IP-packet with a new IP header. Figure 5.2.1 shows ESP and AH in transport and tunnel mode in IPv4. IKE is the key exchange protocol in IPSec. The protocol implements the Oakley key exchange and Skeme key exchange inside the Internet Security Association and Key Management Protocol (ISAKMP). IKE negotiates IPSec security associations (SA). It provides authentication of IPSec peers and negotiates IPSec keys.
IPSec is an standard.

Implementation: http://snad.ncsl.nist.gov/cerberus/

5.2.2 Internet Key Association and key Management Protocol (ISAKMP)

Stallings [15] describes ISAKMP/Oakley as the default automated key management protocol for IPSec. Internet Key Association and Key Management Protocol (ISAKMP) uses the Oakley key determination protocol for key exchange. The key management consists of determination and distribution of secret keys. Oakley is a key exchange protocol based on the Diffie-Hellman



Figur 5.2.1: ESP and AH in transport and tunnel mode

algorithm. ISAKMP defines procedures and packet formats to establish, negotiate, modify and delete Security Associations (SA). There exist five different types of exchange in ISAKMP: Base Exchange, Identity Protection Exchange, Authentication Only Exchange, Aggressive Exchange and Informational Exchange. In Base Exchange key exchange and authentication are transmitted together. Identity Protection Exchange extends Base Exchange to protect the user's identities. Authentication Only Exchange performs mutual authentication without key exchange. Aggressive Exchange does not provide protection of identities and minimizes the number of exchanges. Informational Exchange provides a one-way transmission of info for SA management.

Implementation: http://web.mit.edu/network/isakmp/dodkmp.html

5.2.3 Simple Key-Management for Internet Protocols (SKIP)

Aziz et al [3] describes SKIP as a key management protocol designed to work with the IP Security protocols AH and ESP. The protocol is, in difference from the majority of key management protocols, not session oriented. It is therefore a more appropriate protocol to use together with IPv4 or IPv6 since they are session-less datagram oriented protocols. Using authenticating RSA keys to provide authenticity and privacy for a datagram protocol such as IP may be a problem with session-oriented key management protocols. One way to solve this problem is by implementing a pseudo-session layer

underneath IP, which is sessionless. However, such a solution is complicated and requires too many operations each time a new session key has to be made. SKIP avoids these problems. It is a very simple, efficient, and statless key management protocol that also as Aziz et al [3] says:

> ...provides straightforward and scalable solutions to permit dynamic reroruting of protected IP traffic through alternate encrypting intermediate nodes for crash-recovery, fail-over, and load-balancing scenarios.

SKIP uses two different keys for authentication of packets (A_kp) and encryption of packets (E_kp). They both origin from a third key, which are retrieved from a fourth key that is a shared session key.

Implementation: http://www.skip-vpn.org/usersguide.html
http://cypherpunks.venona.com/date/1995/08/msg00572.html

5.2.4 Network Layer Security Protocol (NLSP)

NLSP is a network layer protocol that provides security services. It is defined by ISO 11577 and is a international standard. Services provided by NLSP are:

- Peer entity authentication
- Data origin authentication
- Access control
- Connection confidentiality
- Connectionless confidentiality
- Traffic flow confidentiality
- Connection integrity without recovery
- Connectionless integrity

Both cryptographic protection between End Systems and between Intermediate Systems located at the borders of security domains, are supported by NLSP.

5.2.5 Security Protocol Layer 3 (SP3)

NIST's network layer security protocol. Much the same as NLSP.

5.3 Transport Layer

Protocols operating at the transport layer do not have to involve the Operating System (OS) during implementation. Instead of encrypting each packet individually, the packets are encryptes per flow.

5.3.1 Secure Socket Layer (SSL)/Transport layer security (TLS)

SSL is the basis for IETF's standard TLS. Petersen and Davie [10] describe the main problem TLS was meant to solve. Transactions on the Web need some level of security. The security requirements differ from service to service. Basically the need for privacy, integrity and authentication must be satisfied. The designers of the protocol bulit a general-purpose protocol that sits between the application layer and the transport layer. The problems to solve were not specific for Web transaction, but important for other services too. The SSL protocol stack is illustrated in figure 5.3.1. Stallings [15] writes that SSL/TLS is a protocol that uses TCP to estabilsh a secure end-to-end service. It consists of two layers of protocols; the SSL Record Protocol on top of TCP, and SSL Handshake Protocol, SSL Change Cipher Spes protocol and SSL Alert Protocol on top of the record protocol.
The Record Protocol provides two services for a SSL connection; confidentiality and message integrity. The record protocol operates by having the application data fragmented. The fragmented data gets compressed (optional) before a MAC is added to it. The compressed data with its MAC gets encrypted and a SSL header is added.
The other layer of protocols in SSL consists of three different protocols. One of them is the handshake protocol. The handshake protocol consists is used to negotiate parameters of the communication. The communicating parties exchange messages through four fases:

- Which security mechanisms do the communicating parties support?
- Server authentication and key exchange
- Client authentication and key exchange
- Finish

The handshake protocol defines a shared secret key that is used for conventional encryption and to form a Message Authentication Code (MAC).
The SSL Change Cipher Protocol consists of one message (1 byte) which purpose is to update the cipher suite used in the connection. The last protocol on this layer is the Alert protocol. The Alert protocol is used to convey SSL-related alerts to the peer entity.

APIs that implement SSL/TLS:

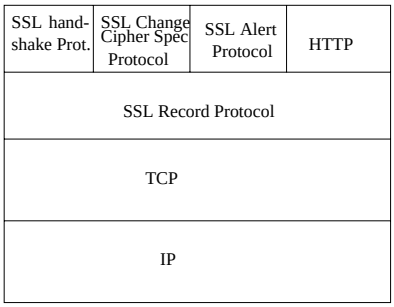


Figure 5.3.1: SSL Protocol Stack

- OpenSSL
- Mozilla's NSS SSL/TLS API
- PKCS-11 from RSA
- CDSA ("Common Data Security Architecture")
- Certicom's SSL Plus
- RSAs BSAFE BHAPI (not open source)

SSL/TLS is a standard.

Implementation:http://www.vonnieda.org/SSLv3/
http://www.openssl.org/docs/

5.3.2 Secure Shell Transport Layer Protocol (SSH)

Ylonen et al [17] gives a description of the SSH transport layer protocol. The SSH-transport layer protocol provides secure remote login and other secure network services over an insecure network. The protocol typically runs over TCP/IP. Services provided by SSH are strong encryption, server authentication, and integrity protection. The two parts of the communication negotiate about key exchange method, asymmetric encryption algorithm, symmetric encryption algorithm, message authentication algorithm and hash algorithm. Since SSH does not provide user authentication, a higher level protocol must be used for that. If transmission errors occurs a SSH-connection will terminate. It is therefore important that the underlying transport protects against such errors.
When establishing a SSH connection both sides start sending identification strings to each other. Then the different negotiations take place. Both sides send a list with algorithms they support. The first algorithm in the list is the one preffered by the sender. All algorithms are negotiated during key

exchange. Key exchange method is Diffie-Hellman. The key exchange gives two outputs: a shared secret key K and a exchange hash H. Encryption and authentication keys are derived from these. A service is then requested by the client. The service is identified by name. ssh-userauth and ssh-connection is the available names reserved.
SSH derives a unique session identificator that may be used by higher level protocols. For example an authentication protocol like Kerberos.

Implementation: http://www.ssh.com/products/ssh/

5.3.3 Private Communication Technology (PCT)

Microsoft's version of SSL. Existed from 1995, but only supported in Microsoft products.

5.3.4 (NSA/NIST Security Protocol Layer 4 (SP4)

SP4 is the transport layer security protocol in NIST Secure Data Network System (SDNS) project. SDNS implements computer to computer communication security for distributed applications. Dinkel et al [5] gives a description of SP4, a secure data network system transport protocol. SP4 provides end-to-end reliable transparent data communications with confidentiality and integrity. One reason for implementing network security at layer 4 is the independence of network technology. Layer 4 is also the fist plave in the OSI architecture where reliable end-to-end connections are established. SP4 encapsulates information. If intergrity is requested a checksum is added, if confidentiality is requested the information and the checksum is encrypted. There are two options in SP4: SP4-E and SP4-C. SP4-E stands for end-to-end SP4 protection, and SP4-C stands for connection-oriented SP4 protection. SP4-C can only be provided when a connection-oriented transport service is available.

5.3.5 Transport Layer Security Protocol (TLSP)

Together with SP4, TLSP represents one of the first proposed standards for network security at the transport layer. The protocol is quite similar to SP4 except from the removal of sequence number. TLSP is also an end-to-end encryption protocol and is centered at the bottom of layer 4, just as SP4. TLSP is a ISO standard.

5.4 Application Layer

Application-layer protocols are embedded within the application. The great advantage with such protocols are that the service can be tailord to the specific application. For M3Ci this may be a solution for applications that for example requires a payment system. It is not always a benefit that all the services ever needed by an application is provided in the infrastructure. There may be many different requirements among them that special requests have to be implementet in the application itself. The next subsections will present some application-layered protocols.

5.4.1 Secure Shell Authentication Protocol (SSH)

SSH Authentication Protocol is described by Ylonen et al [16]. The SSH protocol, in general, is described in section 6.5.2. The SSH Authentication protocol is intended to run on top of SSH Transport Layer Protocol. It provides user authentication, but assumes that underlying protocols takes care of integrity and confidentiality protection. The authentication mechanism is negotiated and the protocol supports multiple mechanisms. The only required authentication method is public key authentication.

Implementation: http://www.ssh.com/products/ssh/

5.4.2 Secure HyperText Transmission Protocol (S-HTTP)

The HTTP extension S-HTTP supports sending data securely over the World Wide Web. Just as SSL does. While SSL is designed to establish a secure connection between two communicating parties, S-HTTP is designed to secure individual messages. Another difference between the two protocols is the authentication part. S-HTTP supports user-authentication, while only the server is authenticated using SSL. The two protocols may be used together.
Shostack [14] gives an overview of S-HTTP. He says that S-HTTP provides mechanisms for integrity, confidentiality and authentication. The messages are encapsulated as encryption, signing or MAC based authentication. The secure version of HTTP supports RSA, in-band, out-of-band and kerberos key exchange. The client and the server negotiate which cryptographic enhancements to use. The certificate types used in S-HTTP could be extended from PKCS-6 or X.509 (see section 6.4.10). Key exchange algorithms can be Outband, Inband, RSA, or Kerberos. Message digest algorithms used in S-HTTP can be 'RSA-MD2', 'RSA-MD5' or 'NIST-SHS'.
S-HTTP is a standard.

Implementation: http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/shttp/

5.4.3 Secure Electronic Transactions (SET)

Stallings [15] says that SET is designed to protect creditcard transactions on the Internet. It is a collection of security protocols and is not itself a payment system.

SET provides three services:

- A secure communications channel
- Trust by the use of X.509v3 cerificates
- Privacy

SET provides confidentiality by using symmetric encryption (DES). It ensures integrity of data by using RSA digital signatures with SHA-1 hash codes. Authentication of cardholder account is done by X.509v3 digital cerificates with RSA signatures. Authentication of the merchant, who the cardholder pays to is also done by X.509v3 certificates. The participants in a SET system are the cardholder, the merchant, the issuer, the acquirer, the payment gateway and the Certification Authority (CA). An important innovation introduced by SET is the dual signature. The dual signature gives extra protection to the customer in linking two different messages for two different recipients. By doing so, none of the recipients are able to change the content of the message.

Information and links to products using SET: http://www.setco.org/

5.4.4 Secure/Multipurpose Internet Mail Extension (S/MIME)

Stallings [15] describes S/MIME as a security extension to the existing electronic mail standard MIME. S/MIME guarantees secure transmission, storage, authentication and forwarding of secret data at the application layer. It provides enveloped data (encrypted content), signed data (digital signature) and clear-signed data.
In S/MIME there are some cryptographic algorithms that must be used and some that are recommended to use. The Digital Signature Standard (DSS) is the preferred algorithm for digital signatures. Diffie-Hellman is used for encrypting session keys. RSA can be used for both signatures and key encryption. As hash function the 160-bit SHA is recommended, while 128-bit MD5 is required. The recommended algorithm for message encryption is

tripple-DES, but it 40-bit RC2 is required.
S/MIME is a standard.

`Implementation:http://www.phaos.com/e_security/dl_smime.html`

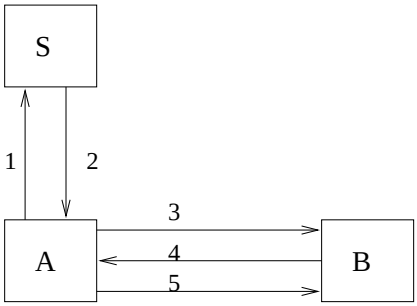
5.4.5 Digital Signature Standard (DSS)

Stallings [15] writes that DSS uses the Digital Signature Algorithm (DSA) to provide a digital signature. DSA makes use of asymmetric encryption, it uses the private key to sign a message and the public key to verify a signature. While the public key may be distributed on web and be entirely public, the private key has to be kept secret. All the security relays on the private key being kept secret. A hash function is used to generate a signature. The hash code and a random number is the inputs to the signature function. The signature function also depends on the senders private key and a global public key. The results from the signature function is a signature consisting of two labels. The receiver generates the hash code from the message. The message and the signature are then input to a verification function. The global public key and the sender's public key, paired with his private key, is also required. If the output of the verification function is the same as one of the parts the signature consisted of the signature is valid.
FIPS [6] says that the DSA is intended for use in electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage, amd other applications which require data integrity assurance and data origin authentication.

Implementation: http://www.itl.nist.gov/fipspubs/fip186.htm

5.4.6 Needham-Schroeder Protocol

Gollmann [7] describes the key transport protocol Needham-Schroeder. Two communicating parties A and B both share a secret key with a server S. The purpose of the protocol is for them to obtain their session key from S. the protocol uses a symmetric cipher for encryption. Replay-attacks are prevented by using nonces (random numbers). Figure 5.4.6 shows how the protocol works. A first contacts S with a message containing A's identity, B's identity and a nonce N1. This message tells the server that A wants to communicate with B. The server S replays with a message encrypted with the session key A and S share. The encrypted message contains the nonce N1, B's identity, the key A and B are going to share and the same key together with A's



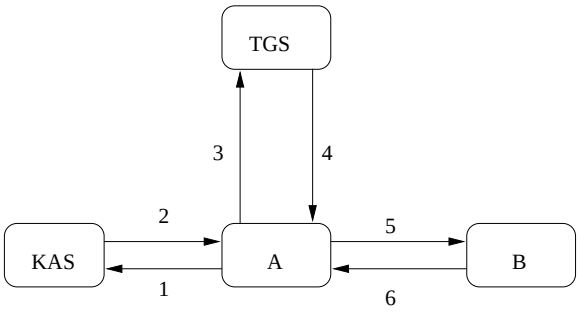
Figur 5.4.6: Needham-Schroeder key transport protocol

identity encrypted with the session key S and B shares. A sendes this last part of the message directly to B who answers with a nonce N2 encrypted with the key A and B are going to share. A responds to B with a encrypted versjon of (N2 - 1). At this time the key is transported and A and B may start to communicate securely.
Stallings [15] points out the possibility for impersonation of A in the Needham-Schroeder protocol. The session key sent from A to B encrypted with the key B shares with S may be catched by an opponent X. X can then send messages to B using this session key. Denning solved this problem by adding timestamps in the second and third step of the protocol. Adding timestamps will however present a new problem: the need of synchronized clocks. To avoid synchronized clocks Neuman introduced his protocol with expiration time added to the origibal Neuman-Schroeder protocol. The expiration time makes it possible to start a new session without communicating with the server first (if the time has not expired).

Implementation:`http://www.compapp.dcu.ie/~cdunne.mect1/security/project/user_guide.html`

5.4.7 Kerberos

Gollmann [7] and Stallings [15] both gives a description about how the Kerberos authentication service works. Kerberos is a network protocol that provide secure authentication of client and server using symmetric encryption. The protocol uses strong cryptography to make it possible for a client and a server to communicate across an insecure network.
Gollmann [7] gives a description of Kerberos. The Kerberos system was developed at MIT in the 1980s and has it's origin i the Needham-Schroeder protocol. The Needham-Schroeder protocol transports keys. It uses a symmetric cipher for encryption (e.g., DES) and authenticates users with passwords not transmitted i clear over a network.
Figure 5.4.7 illustrates the steps in a mutual authentication of the user A and the server B. The first step involves the user and a Kerberos Authentication



Figur 5.4.7: Kerberos

Server (KAS). A contacts KAS with a message consisting of two identifications, the Identification of A and the identification of the Ticket Granting Server (TGS). The TGS will be contacted by A as soon as he receives the ticket that he shares with the TGS from the KAS. The message also consists of a lifetime stamp L1 and a nonce N1. A nonce is a randomly picked number to prevent replay attacks. The second message sent during Kerberos authentication is from the KAS to user A. The KAS sends an encrypted version of the identificator TGS, a session key K shared by TGS and A, a ticket shared between A and TGS, L1 and N1. The KAS has now authenticated A and given him the opportunity to contact TGS. A needs a Ticket Granting Ticket (TGT) from TGS to communicate securely with B. The third messages goes from A to TGS. A sends an identification of himself and one of the server B which he wants to communicate with. He also sends a new lifteime stamp L2, a nonce N2, the ticket he recieves from KAS and an encrypted message with his own identificator and a creation time of tickets/authentication T1. A encrypted the message with the key he recieved from KAS and which he shares with TGS. TGS replays with a encrypted message, with the same key, which consists of the identificator of B, a session key that A and B shares, a ticket A shall use when contacting B and the lifetime stamp and nonce he got from A. A is now able to communicate with be in a secret way, and he sends an encrypted message with his identificator and the creation time of tickets/authentication together with the ticket. B replays by encrypting the creation time stamp with the session key to show A that he is who he claims to be. A is now authenticated.

The protocol is freely available at MIT: http://web.mit.edu/kerberos/www/

Kerberos Library:
http://web.mit.edu/macdev/Development/MITKerberos/MITKerberosLib/Common/Documentatio

5.4.8 Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) was develoed by Paul Zimmermann. PGP provides authentication and confidentiality services and has become a popular security application protocol. It is free, and available worldwide. Zimmermann used the best cryptographic algorithms available and PGP has a wide range of applicability. Another benefit is that it is neither developed nor controlled by a standard organization. Stallings [15] describes the 5 services provided by PGP:

- Digital signature using DSS/SHA or RSA/SHA
- Message encryption using CAST or IDEA or Three-Key Triple DES with Diffie-Hellman or RSA
- Compression using ZIP
- Email compatibility using Radix 64-conversion to convert to ASCII string
- Segmentation and reassembly

Authentication is provided by a digital signature. Confidentiality is provided by encrypting the message. PGP where developed for e-mail security, but is now used for other applications too.
PGP is a standard.

Implementation of PGP: http://www.pgpi.org/

5.4.9 Pretty Good Privacy Library (PGPLIB)

PGPLib is a library based on Pretty Good Privacy (PGP). There is no documentation about the library available. It was written because the developer needed it himself. PGPLib let you generate and manipulate PGP packets without running PGP.
The following types of PGP packets can be generated:

- Data can be signed with a private key
- Data can be encrypted with a public key
- Data encrypted with your public key can be decrypted
- You can verify signatures on public keys and on buffers (files)
- Conventional packets can be encrypted and decrypted (IDEA)
- A buffer or file can be (de)armored into a buffer or file

- UserID packets are read and written in a varity of formats
- Literate packages can be created from files, or from buffers and files can be created from literate packets
- Keys can be obtained from a database or by parsing keyrings
- You can maintain a PGP public-key database

Information:

`http://www.openbsd.org/2.7_packages/i386/PGPlib.tgz-long.html)`

Implementation of PGPLib: http://www.cam.org/ droujav/pgp/pgplib.html

5.4.10 Privacy Enhanced Mail (PEM)

Petersen and Davie [10] describes PEM as a mechanism for encryption, authentication and integrity of e-mail messages. A RSA public key mechanism is used for authentication and encryption. It is therefore necessary with a mechanism to distribute public keys. PEM uses a hierarchical tree structure with Certification Authorities (CAs) to distribute certificates. This method of distributing certificates scales much better than one centralized authority who have to sign all cerificates. One CA may sign a sertificate on behalf of another. The problem with this solution is the delegation of trust from one CA to another.
PEM is a standard.

Implementation of RIPEM: http://camtech2000.net/Pages/Downloads.html

5.4.11 X.509

Peterson and Davie [10] defines a certificate as

> a special type of a digitally signed document that says "I certify that the public key in this document belongs to the entity named in this document,signed X". X could be anyone with a public key, but commonly X would be the Certfication Authority (CA).

Stallings [15] gives a description of the ITU standard for digital signatures, X.509. The certificates contain the public key of a user and is signed with the private key of a CA. The X.509 certificates are used in protocols as

IPSec, SSL/TLS, S/MIME and SET. X.509 is based on the use of public key cryptography (recommended algorithm is RSA) and digital signatures.
These are the fields in the X.509 certificate:

- Version, tells us the version of the certificate. Default value is version 1. There also exist version 2 and a version 3
- Serial Number; integer which are unique within this issuing CA
- Signature Algorithm Identifier; identifies the algorithm used for signing
- Issuer Name; X.509 name of CA
- Period of Validity; first and last date this certificate is valid
- Subject Name; name of the user
- Subject's Public Key Information; public key and an identifier of the algorithm to use it with
- Issuer Unique Identifier
- Subject Unique Identifier
- Extensions; extension fields added in version 3
- Signature; the hashcode of all the other fields encrypted with the CA's private key.

Stallings [15] also shows the standard notation to defining the certificate:
CA«A» = CA {V, SN, AI, CA, Ta, A, Ap}
CA is the issuer and A is the user. The issuer signs the certificate with his private key. If the user knows CA's public key he can verify that the certificate is signed by the CA. To prove that you are the subject mentioned in the certificate, you need something that shows that you have the private key corresponding to the public key in the certificate. How many CAs needed depend on how many users there are. Many users need many CAs. The CAs can for example be organized in a hierachy where the CA over X has signed X's certificate.

Information: http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-howto.html

6 Recommendation

There exists many APIs and probably even more security protocols. It is therefore difficult to choose the best suited mix for our usage. GSS-API is

the security which are most generic. It is also the best suited API for an application programmer to use. The APi supports alle the security services we need in our platform and works well together with lower level cryptographic APIs (CAPIs). The documentation around GSS-API is good. There exists many RFCs about it, and some of them consider protocols and mechanisms that may work together with GSS-API. Both public- and private key mechanisms are described. Kerberos version 5 GSS-API mechanism (RFC 1964) is already a eatblished solution. As a representative for the public-key mechanisms SPKM (Simple Public-Key mechanism) is documented. While Kerberos supports authentication and not much more, SPKM provides both unilateral and mutual authentication, digital signatures,

7 What more do we need?

There exist many APIs for supporting security in a system, but using only one is not enough. There are several layers of security APIs wich has to be taken into account. An lower-level API that takes care of encryption need to be used together with a higher-level API if authentication and integrity are required. The same thing is the case when it comes to protocols. What we need is some kind of library consisting of the perfect mechanisms for our project. Such a library does not exist, of course, but by connecting different mechanisms that provides the generic security services we need, we can get a perfectly good solution.

References

[1] Adams, C., *Independent Data Unit Protection Generic Security Serivice Application Program Interface (IDUP-GSS-API)*, Network Working Group, RFC 2479, Entrust Technologies December 1998

[2] Amit, A., *Security and security API in Java*, 1999

[3] Aziz, A. et al, *Simple Key-Management for Internet Protocols (SKIP)* 1997

[4] Cisco Tech note, *PPP Authentication using the PPP*, `http://www.cisco.com/warp/public/131/ppp_callin_hostname.html`

[5] Dinkel, C. et al, *NISTIR 90-4228 A Secure Data Network System Tranport Protocol Interoperability Demonstration Project*

[6] Federal Information Processing Standards Publications (FIPS PUB), *Digital Signature Standard (DSS)*, May 1994

[7] Gollmann, D., *Computer security*, Wiley, 1999

[8] Linn, j., *Generic Security Service Application Program Interface*, Network working group, RFC 2743, RSA Laboratories January 2000

[9] NSA Cross Organization CAPI Team, *Security Service API: Cryptographic API Recommendation Second Edition*, NSA, 1996

[10] Petersen, L. L. & Davie, S. B., *Computer Networks* Morgan Kaufman, 2000

[11] Rogaway, P., *Generic Cryptographic Service Application Program Interface* IBM, 1994

[12] Ryotov, T. & Neuman C., *Access Control Framework for Distributed Applications* CAT Working Group, 2000

[13] Ryotov, T. et al, *Generic Authorization and Access control Application Program interface C-bindings* CAT Working Group, 2000

[14] Shostack, A., *An overview of SHTTP* 1995

[15] Stallings, W., *Cryptography and network security*, Prentice hall, 2nd edition 1999

[16] Ylonen, T. et al, *SSH Authentication Protocol*, Network Working Group, Internet Draft, 1999

[17] Ylonen, T. et al, *SSH Transport Layer Protocol*, Network Working Group, Internet Draft, 2001