1

PLASMA
Platform for Secure Multimedia Applications

A. Krannig
Fraunhofer–Institut für Graphische Datenverarbeitung
Wilhelminenstr. 7
64283 Darmstadt, Germany
email: krannig@igd.fhg.de

Abstract

PLASMA, a platform for secure multimedia applications, allows secure communications within public networks and is particularly suited for transmitting compound multimedia documents.

Using PLASMA, it is possible to negotiate the cryptographic protocols to be applied before starting the communication proper. This allows communications between domains using different security policies which, in addition, can adapt to the media–specific structures of the data to be transmitted.

This paper describes the software design of PLASMA which has already proven itself to be powerful and flexible during the ongoing implementation phase. The PLASMA system was created by the Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt in cooperation with and funded by De.Te.Berkom, Berlin.

Keywords

PLASMA, Multimedia, Security, Cryptography, Object-Orientation

1 INTRODUCTION

Significant progress has occurred in the fields of multimedia telecommunications and information services. For the most part, the absolute necessity of securing transactions and communications was not given adequate considerations since other, mostly technical, problems were predominant at the time. Now that these systems are leaving the research community and enter mainstream commercial applications, the urgent need for securing communications becomes more and more obvious. To allay this need, the PLASMA project, a security platform that offers the necessary cryptographic services, was initiated.

P. Horster (ed.), *Communications and Multimedia Security II*

PLASMA represents a security platform which all present or imaginable future multimedia applications can use to communicate safely on public computer networks. To achieve this, the PLASMA interface for applications must be as transparent as possible to remove the application program from the unnecessary specifics and often arcane cryptographic protocols and related details. Furthermore, this interface must allow applications written in conventional languages as well as modern object–oriented applications to access the security layer with similar ease. The AU–API * is modeled after the GSS–API (Linn, 1993), (Wray, 1993) with regard to ease of use and limiting the complexity visible to the application. In addition, PLASMA uses the concept of a `Filter` — which is used to transform the application–specific document structure of application data into generic PLASMA categories — as well as a `Delegate` which can be imagined as a mailbox for PLASMA inside the application where PLASMA can deposit data for a remote PLASMA session as well as retrieve data. `Filter` as well as `Delegate` are modules whose interface is specified by PLASMA but are implemented inside the application. These concepts allow a flexible approach to document transformation and transmission.

In a similar vein, the security platform should be able to support several cryptographic technologies in a future development phase. This necessitates a technology independent API (TU–API).

The application documents are operated upon according to rules given in a "Kooperations–Sicherheits–Politik (KSP)" (cooperative security policy). The KSP is computed from the security policies of the communicating partners. These are exchanged between the partners before secure communications commence.

If, based on the security policies, an agreement on compatible security mechanisms is possible, communications between domains with different security policies becomes possible.

Assuming both parties have PLASMA deployed and are able to reach an agreement on which protocols and algorithms to be used, it becomes sensible to differentiate between multimedia data types and use appropriate cryptographic procedures based on the data categories. For example, there are scrambling algorithms for video streams as well as selective encryption techniques for images and video data which will merely reduce the quality of the images. Copyright protection for image data becomes interesting since modern computer graphics allows creative processes impossible without the aid of computers. It is also reasonable to assume that digital signatures for images are calculated differently from textual data so as to allow changes in single data bits. This does not constitute a visible change in the image, but would invalidate a digital signature. In conclusion, it is to be assumed that for multimedia data types such as audio, video or text there is be a plethora of new cryptographic techniques which currently are still under development. Furthermore, there are problems associated with multimedia data which were never considered relevant for textual data. PLASMA has the technology to differentiate between media types of data and operate on the data accordingly.

This makes PLASMA the only application–centric security platform available since its **Filters** can interpret application–specific documents and return these documents to their original format after the secure communications process has taken place in a transparent fashion.

The PLASMA security platform differs significantly from the approaches previously taken in its independence from communications mechanisms, i.e. while other security

*Anwendungsunabhängige Schnittstelle or Application independent API

platforms are grafted onto a lower (network) layer, PLASMA relies upon whatever communications mechanism the application may choose to use.

2 THE FUNCTIONALITY OF PLASMA

The requirements posed to PLASMA are constituted by two main components. First, there are the threats posed to telecommunications once conducted over open computer networks, second there are considerations coming into play regarding media specific cryptographic treatment of electronic data which are only making secure multimedia communications feasible in the first place. Based on these two broad requirements, there are further demands for

- **Generic Services**
  The cryptographic mechanisms such as DES, RSA etc. can be used in such a fashion as to treat multimedia data specific to their media type. For a description of their characteristics see Schneier (1996).
- **application independence**
  PLASMA is application independent, so multimedia applications can easily be integrated with PLASMA (e.g. conferencing systems, multimedia mail applications ...)
- **technology independence**
  PLASMA is technology independent, so different cryptographic technologies (e.g. software, hardware, smartcard implementations) as well as different security platforms (e.g. SecuDe (Schneider, 1993), PGP (Zimmermann, 1993), Docrypt) can be used.
- **communication independence**
  PLASMA is not bound to a communications platform since the application handles all communications as it sees fit.
- **domain–transcending security**
  Facilitated by the concepts of security policies, their interpretation and use of a cooperative security policy, the communicating parties have the technology for secure communications beyond their own domains. (For details see Gehrke and Koch (1995)).
- **application–centric**
  The PLASMA model is centrered on the application document. This allows PLASMA to differentiate between application media types.
- **media–specific cryptographic treatment of data**
  PLASMA recognizes different media types by using a `Filter` and can treat data types differently applying different security protocols.

3 THE PLASMA ARCHITECTURE

A rough outline of the PLASMA architecture is presented in figure 1. An application accesses the system via the **Delegate** and **Filter** components. It is the responsibility of the Filter to transform application documents into a well–defined canonic form which allows PLASMA to access the specific media components of the document. These documents are then operated upon by PLASMA according to the **Cooperation Security Policy** (KSP). The KSP is calculated from the security policies of the communications partners.

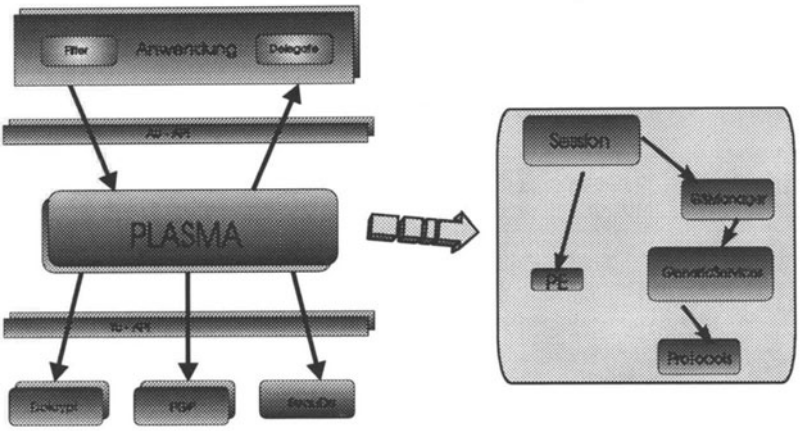

Figure 1 The PLASMA architecture

These are exchanged prior to the communications proper. The actual sending process of messages between users is delegated by PLASMA to the **Delegate** of the application.

The object–oriented approach is justified not only by fashion but by the demands for technology independence and media–specific treatment of data objects.

If one considers the media as objects of different classes such as `Text`, `Image`, etc., it is easy to operate on these objects using different cryptographic algorithms; the specific objects have internal formatting specifications which allow them to decide which elements of the internal representation have to be encoded and encrypted and which should not get modified. PLASMA was designed using OMT, an object–oriented methodology by (Rumbaugh et al., 1988) and implemented using the C++ programming language.

3.1 The object–oriented design approach

The demand for technology independence is the prime reason for the object–oriented design. Using this technology it becomes feasible to employ a simple yet elegant design which allows one to ignore differences in technologies as long as possible whereas conventional programming techniques would require tedious and cumbersome distinctions between technologies at every step of the way. This will be further elaborated on in the following pages. The OMT method (Rumbaugh et al., 1988) used in the design process specifies the application design in terms of three models:

- **The static model**
  This model specifies the static relationships between objects such as aggregations and associations; it also creates an inheritance hierarchy of classes by factoring out common

behavior. The static model deals primarily with object and class diagrams as tools for visualization.

- **The dynamic model**
  This model reflects the dynamic behavior of the system during runtime. It shows up the interactions between the objects, such as which messages are passed between objects and how these objects react to the messages. Objects have internal states. State diagrams describe the ways in which an object reacts to messages depending on it internal state. As a tool for visualizing these interactions, state diagrams and interaction diagrams are used.
- **The functional model**
  This model allows the designer to model input and output behavior of objects. It is useful to define external interfaces of objects and to verify the two preceding models. It is developed largely independently from the static and dynamic model but has to be synchronized with these models.

In our experience, having detailed and comprehensive static and dynamic models virtually obsoletes the functional model. This is especially true if, as in this case, most semantic actions cannot be captured completely by a functional model in the first place. We therefore describe only the first two models in this paper.

In hindsight however, OMT has proven to be a valuable tool for designing a complex piece of software. Having a common vocabulary and a guideline for structuring the development process itself has allowed us to strike a balance between the design and implementation viewpoints.

4 THE STATIC MODEL OF PLASMA

At first the different classes and objects necessary for accomplishing the tasks at hand have to be carved out.

First of all, a storage space for the user's sensitive data has to be provided. This storage must be protected from tampering, even by system supervisors. The best implementation for such storage is in the form of a **SmartCard**, although it is possible to provide a cryptographically protected software implementation. In the PLASMA project, this is called a **Personal Environment** and is implemented using the `PE` class hierarchy.

Furthermore, `FSP` and `KSP` objects are necessary which will hold the user's security policies as well as the negotiated coordinated security policies for communications.

`GenericServices` supply cryptographic services using different algorithms or cryptographic protocols. This facilitates the selection of compatible mechanisms between communicating partners and to select media–specific algorithms.

A `Packer` encodes an object together with its aggregate subobjects (members) into a "flat" character string on the sender side and decodes it back into an object hierarchy on the receiving side. A `Packer` must be platform–independent.

The `Filter` takes an application document and generates a `Container` from it which contains the elements of the document grouped by media types in the form of a list. In order to recreate the original document on the receiver's end, the `Filter` creates a `Script`. This `Script` describes the receiver's `Filter` how the `Container` is to be interpreted to recreate the original document. The `Script` class inherits its behavior from the `Data` class and is in this respect similar to the remaining data types such as `Audio` and `Text`.

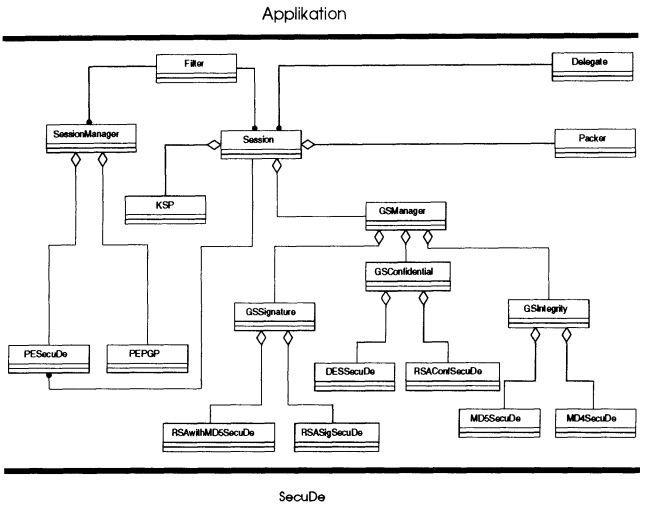

Figure 2 Static object model overview

A `Session`–Object maintains an active communications session in PLASMA. For each ongoing communication between the user and a remote partner, a `Session` with its aggregate subobjects is maintained.

The `Delegate` is the "mailbox" of the application into which PLASMA deposits the cryptographically treated document. PLASMA also retrieves the encrypted document from the mailbox and reverts the cryptographic treatment.

Furthermore there exists a `GSManager` and the `GenericServices` which aggregate the `Protocol` objects. The `GenericServices` are maintained by the `GSManager`.

Finally, the `SessionManager` maintains the different sessions of the user.

Figure 2 illustrates the static relationships between the PLASMA objects.

The inheritance hierarchies of the PLASMA *classes*

The `PE` *classes* The interface to the **personal environment** is provided by the `PE` hierarchy. Derived from this abstract interface class are the classes `PESecuDe` and `PEPGP` which provide the technology–dependent implementation. Only one instance of each of the technology–dependent derived classes exists at any given time, no matter how many sessions the user has active. The `SessionManager` creates a technology–dependent `PE`, e.g. a `PESecuDe`. The `Session`–Object will access this object only by means of its technology–independent superclass interface. This superclass (`PE`) organizes access to the technology–dependent `PE`.

The `GenericService` *classes* The `GSConfidential`, `GSSignature` and `GSIntegrity` classes inherit from the `GenericService` class; they constitute the generic services which

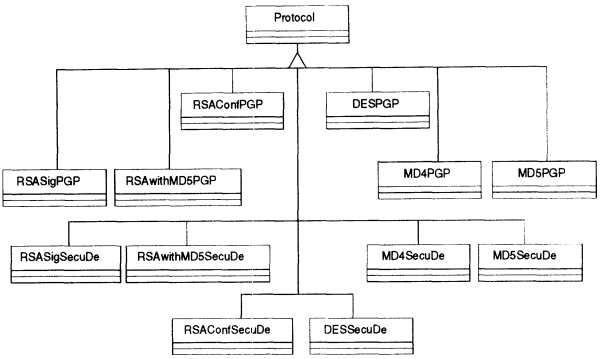

Figure 3 The `Protocol` inheritance hierarchy

aggregate the cryptographic service protocols providing "Integrity", "Confidentiality" and "Liability".

The `Protocol` *classes* The cryptographic protocols access the actual cryptographic algorithms of the security platform providing the implementation such as **DES**, **RSA**, **MD5** etc. which will operate on the data. All protocols inherit from the class `Protocol` and are accessed by the same methods.

The `Codable` *classes* All classes whose instances have to be sent to the remote session inherit from the class `Codable`. To facilitate a common interface to the codable classes, the `Codable` class defines the necessary methods. These will be overloaded and adapted as necessary by the subsequent classes in the hierarchy. They provide the methods to allow each object to en-/decode itself between a binary representation and a flat memory block format.

The `Data` *classes* The `Data` class is a superclass for those PLASMA classes holding the multimedia data types of an application document. The protocols aggregated by the generic services communicate with these objects by means of the `Data` interface. This allows cryptographic protocols to access the data contained in the subclass objects as well as to attach additional data required for digital signatures and integrity-related data.

5 THE DYNAMIC MODEL OF PLASMA

The dynamic model is divided into scenarios which will reflect the behavior of objects in a given situation.

The dynamic modeling process works out the object behavior during runtime. These are some of the questions a developer must pose himself: How do the objects interact with one another and which messages do they exchange ? Which event follows which message, in what sequence are the messages exchanged and how do they influence the internal state of the object ?

At first we will describe a rough outline of the session creation scenario which includes authentication of users by the ITU (CCITT) X.509 protocol. Following the outline will be six more detailed sub–scenarios.

The SessionManager generates a FSP instance from a file located in the user's home directory. This FSP object queries the SessionManager for the PlatformID of the communicating partner.

The SessionManager configures the Session depending on this PlatformID which represents the underlying security platform (e.g. SecuDe, PGP). Once the Session is configured and the application has obtained a HANDLE to this specific Session, the application initiates the first round of the authentication protocol with the communications partner.

The sender commences the authentication according to the X.509 protocol. In the course of this process, the public keys, the session key as well as the FSPs of the parties are exchanged.

Once the authentication phase is successfully completed, both sides compute the KSP from their own FSP and the FSP of the partner. Now they are able to communicate in a secure fashion using PLASMA.

Due to space constraints, only the first three of the following scenarios of the dynamic model will be further elaborated upon (these, however, do give an adequate picture of the operating procedures):

1. PLASMA initialization – the SessionManager generates the technology–dependent PEs.
2. The SessionManager creates a Session: the technology–dependent Protocols are generated and associated with the GenericServices.
3. Document processing on sender side.
4. Document processing on receiver side.
5. Ending a Session.
6. Ending PLASMA – the destruction of the SessionManager and the technology–dependent PEs.

5.1 Scenario 1: PLASMA initialization

In the initialization phase, PLASMA creates the SessionManager. For each technology platform, the SessionManager has a reference to a technology–dependent PE.

The SessionManager constructor generates the technology–dependent PEs. The SessionManager therefore also aggregates the technology–dependent PEs and destroys them when appropriate. The instances of SessionManager, PESecuDe, PEPGP etc. are in existence before the first Session is created.

All Sessions based on SecuDe will access the same PESecuDe, sessions based on PGP will access the same PEPGP etc. Every Session is given a reference to the PE upon creation. It is irrelevant to the Session which particular technology this PE is based on since it will access the technology PE like PESecuDe, PEPGP etc. only through the PE superclass interface. The Sessions are merely responsible for opening and closing the PE. The PE will keep track of how many active Sessions are accessing it in an attribute count. Once there are no more active Sessions (either closed or deleted), all technology–dependent PEs must have count values of 0; otherwise the SessionManager and user must receive an error message.

A Session references therefore a specific technology–dependent PE without knowing

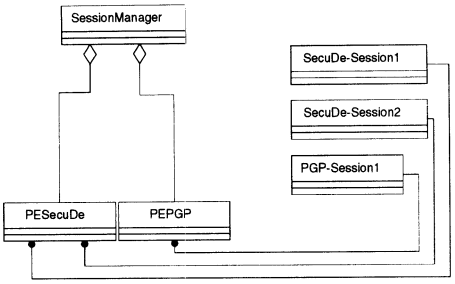

Figure 4 Maintaining `Session` instances

it. The object–oriented design proves its value in this instance by providing a clean, yet flexible, solution.

5.2 Scenario 2: The `SessionManager` creates a `Session`

Overview

The `SessionManager` is the only instance within PLASMA which has knowledge regarding the platform it operates on. No other object needs to have knowledge of the technology, even if they store the local platform ("SecuDe", "PGP" etc.) in an attribute.

Technology dependent objects, such as the `PESecuDe` and the technology–dependent `Protocols` like `DESSecuDe`, `RSAConfSecuDe` etc. are accessed throughout PLASMA only by their superclass interface provided in `PE` and `Protocol` which define a common interface.

To give an illustration, there exists a so–called "`GSManager`–tree". A `GSManager` aggregates different `GenericServices` which offer cryptographic services like "Integrity", "Confidentiality" and "Liability".

Each `GenericService` aggregates technology–dependent `Protocols` which perform the necessary cryptographic operations for the service in question.

It is now important to note that the `GenericService` layer (which is located directly above the technology–dependent `Protocol` layer) can treat these `Protocols` exclusively as **technology–independent**. The `GenericServices` have no knowledge of the underlying technology platform.

If the `SessionManager` creates a `RSAConfSecuDe`, he merely passes the `GSConfidential` a reference to a `Protocol`. This is polymorphism in action.

Note that in PLASMA, cryptographic algorithms are referred to as **Protocols** since, for example, if the application data is encrypted by an **DES** algorithm on the sender side, the protocol for reconstructing the original data from the encrypted object is clearly defined. Therefore cryptographic operations may be viewed as a clearly defined protocol between sender and recipient.

A `SessionManager` can maintain several `Sessions` simultaneously as depicted in figure 4. Each `Session` is configured as shown in figure 2.

Dynamic operations

The application sends PLASMA a message to create a new session. PLASMA passes the appropriate message to the SessionManager. The SessionManager generates a FSP from the file in the user's home directory. Depending on the PlatformID — which may hold values like "SecuDe", "PGP" etc. — the SessionManager now configures the Session.

The technology–dependent objects are already declared as Protocol within the SessionManager.

The SessionManager now creates the GenericServices and passes the constructors the appropriate Protocol instances. The GenericServices can see the cryptographic protocols only as generic Protocols.

Once the GenericServices are created, the SessionManager generates the GSManager. The GSManager may receive an arbitrary number of GenericServices at any time after its own construction.

After this, the Packer and KSP objects are constructed (depending on the platform). Now the manager calls the Session constructor and passes it the references to the previously created objects. The Session constructor opens the PE. Once the Session is created, the SessionManager can pass the GSManager a reference to the newly created Session.

Now the SessionManager has completed the task of creating and opening a Session; the method call returns to the calling application. The application receives a HANDLE to the Session which it has to map internally to the communicating partner.

5.3 Scenario 3: Document processing on sender side

The application passes a multimedia document to the Filter in order to have it cryptographically treated by PLASMA.

The Filter now partitions the document into its media–specific components and generates data objects from it which are stored in a Container. The Filter queries the SessionManager by means of the HANDLE passed in by the application for a reference to the appropriate Session. This Session is then passed the Container by calling the appropriate method. Session passes this message on to the corresponding GSManager method. The GSManager queries the Session for its reference to the KSP if it does not have the cached value by them.

Subsequently, the GSManager retrieves the first data object from the Container. This data object is then queried for its data type, i.e. whether it is a Text, Video, Audio etc. object. Now the GSManager queries the KSP for a resolution for this data type and the first GenericService in its service list on if – and how – to treat the data cryptographically. The KSP also informs the GSManager on which protocols will supply this service. The data object is then passed on to the according GenericService and from there to the proper Protocol. This procedure is performed for each GenericService and for each data object until the Container runs out of data. The GSManager then returns control to the Session which passes the encrypted Container on to the Packer. The Packer transforms the Container and its aggregate objects into a linear character stream and subsequently returns control to Session which then passes the stream on to the Delegate of the application.

The application is now responsible for transferring this cryptographically modified document (which, for its purposes, is a long character string) to the receiver. The method used for this transport layer is entirely up to the application.

Once again the advantages of object–orientation are obvious, this time in the case of the `Codable` class hierarchy. All transforming methods of the `GenericServices` and the `Protocols` as well as the transport-encoding functions of the `Packer` are working with `Codable` base class parameters exclusively. Each image or text object, each data object, each key and each message exchanged during authentication inherits from the class `Codable` and can be treated as such.

6 CLOSING REMARKS

In this paper, the authentication phase as well as the application–independent interface – which allows multiple applications to acquire PLASMA functionality effortlessly – was not given consideration in this paper. The authentication takes place according to the ITU (CCITT) X.509 Authentication Framework standard. The actual exchange of authentication token and generation of token is realized by means of objects within PLASMA. During this authentication phase, the public keys of the users as well as the `SessionKey` and the `FSPs` of the participating partners are exchanged. Within the user's local `FSP` is a description of the cryptographic algorithms the system is supposed to use for each data class. This **Security Policy** might, for example, express that an image shall be encrypted by a simple **XOR** method whereas a text must be encrypted with the **DES** algorithm. Assuming this to be the `FSP` of user A (`FSP_A`), a `FSP` of a communications partner B (`FSP_B`) will contain similar rules. If A and B decide to communicate, they may use algorithms that are in the set intersection of `FSP_A` and `FSP_B`.

Exactly this set intersection of each user's `FSPs` is referred to as the `KSP`. This basis for communications is calculated by means of an interpreter from the formal security policies of the participating partner.

7 OUTLOOK

There are many points currently not addressed within PLASMA such as whether or not to store the calculated `KSP` for two users for future reference when re–initiating communications with the same partner: what happens if one of the `FSPs` forming the basis for calculating this `KSP` is modified ? How can other users be notified that this `FSP` (and therefore the entire `KSP`) is no longer valid? What is a useful maximum lifetime for a `KSP`?

Further points to consider might be the problem of finding out the security platform the potential partner has if the user has never before engaged in communications with that partner before on the PLASMA side. Entering a "compatibility mode" may be an option although it would defeat the multimedia capabilities of PLASMA.

An approach similar to one currently under consideration with regard to public key encryption (centralized storage of public keys in a globally accessible location) may be useful here but would also require a specific cryptographic procedure.

Even after the recent relaxation of laws regarding cryptography in the U.S. and the charges against P. Zimmermann being dropped by the U.S. DoJ, politics may dictate which cryptographic technologies to use. The ongoing political debate on encryption technology

and accessibility for civilian use is, sadly, beyond the scope of this article, despite the tremendous impact it will have on global communications and commerce.

How does PLASMA perform in open, distributed systems ? Can it be used to secure communications in distributed systems ? Can a connection to CORBA, a de–facto standard for open systems be established? With standards either totally absent or inadequate for systems encompassing the whole of the Internet and not just a college campus, a wide range of future research topics that need to be addressed can be seen here.

8 ACKNOWLEDGEMENTS

The author would like to express his gratitude to the following people:

Dr. Eckhard Koch for providing a supportive environment and the freedom to pursue unusual approaches, Mehrdad Jalali for his cooperation and input in the design phase, Stephen Wolthusen for providing a viable implementation and Dr. Michael Gehrke for providing seminal ideas.

9 REFERENCES

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1988) *Object–Oriented Modeling and Design*, Prentice Hall.

Schneier, B. (1996) *Applied Cryptography, 2nd ed.*, Wiley.

Zimmermann, P. et al. (1993) *PGP*

Schneider, W. (1993) *SecuDe: Overview* GMD–TKT Darmstadt.

Linn, J. (1993) *RFC 1508 – Generic Security Service Application Programming Interface*

Wray, J. (1993) *RFC 1509 – Generic Security Service API : C Bindings*

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) *Design Patterns Elements of Reusable Object–Oriented Software* Addison Wesley.

Meyer, B. (1988) *Object–oriented Software Construction* Prentice Hall.

Kaliski, B. S. Jr. (1994) *Cryptoki, A Cryptographic Token Interface* RSA Laboratories

Kaliski, B. S. Jr. (1993) *An Overview of the PKCS Standards* RSA Laboratories

Kaliski, B. S. Jr. (1993) *PKCS # 3: Diffie-Hellman Key Agreement Standard* RSA Laboratories

Kaliski, B. S. Jr. (1993) *PKCS # 3: Private-Key Information Syntax Standard* RSA Laboratories

Gehrke, M. (1994) *Eine Sicherheitsarchitektur für kooperative Umgebungen* Dissertation.

ISO/CCITT (1991) *ISO 9594-1/X.500, The Directory: Overview of Concepts, Models and Services* ISO/CCITT

ISO/CCITT (1991) *ISO 9594-8/X.509, The Directory: Authentication Framework* ISO/CCITT

Gehrke, M. and Koch, E. (1995) *A Security Platform for Future Telecommunication Applications and Services* Proc. of the 6th Joint European Networking Conference.

Vásquez-Gómez, J. (1994) *Multidomain Security* Computers & Security Vol. 13