IST-2000-25350 - SHAMAN

Deliverable Number	D08
Deliverable Title	Intermediate Specification of Security Modules
Date of delivery	
Document Reference	SHA/DOC/GD/WP4/D08/1.1

Contractual Delivery Date	28-Feb-2002
Actual Delivery Date	15-May-2002
Editor	Hans-Jürgen Heinrich, G&D

Participant(s):	G&D, NOK, VOD
Workpackage	WP 4
Est. person months	
Security	Public
Nature	Report
Version	1.1
Total number of pages	62

Abstract:

The document D08 is the intermediate report from SHAMAN WP4. The document derives the requirements for a security module (SM) from the SHAMAN scenarios describing secure heterogeneous network access, personal area networks and personal CAs. It contains a reference model which can be used as a basis for the implementation of a SM. Finally the reference model is compared with current implementations of tamper-resistant devices.

Keyword list: Security module, tamper-resistant device, smartcard

Contributors

Name	Affiliation	Email	Phone
Ertl, Hubert	Giesecke & Devrient (G&D)	Hubert.ertl@de.gi-de.com	+49 89 4119 2796
Heinrich, Hans-Jürgen	Giesecke & Devrient (G&D)	Hans-Juergen.Heinrich@de.gi-de.com	+49 89 4119 2625
Niemi, Valtteri	Nokia Group (NOK)	valtteri.niemi@nokia.com	+35 840 5331438
Sovio, Sampo	Nokia Group (NOK)	Sampo.Sovio@nokia.com	+35 8504837440
Howard, Peter	Vodafone Ltd. (VOD)	peter.howard@vodafone.com	+44 1635 676206
Sondh, Jagjeet	Vodafone Ltd. (VOD)	Jagjeet.Sondh@vodafone.com	+44 1635 682927
Mitchell, Chris	Royal Holloway Univ.	me@chrismitchell.net	+44 1784 443423

Table of contents

Change History

Version number **(SHA_DOC_GD_WP4..)**	**Date**	**Remarks**
D08_0.8	2-May-02	Final Draft for PMC review
D08_1.0	14-May-02	Final Document after PMC review
D08_1.1	15-May-02	Editorials and clarifications on PAN

List of abbreviations

AAA	Authentication, Authorization and Accounting
CA	Certification Authority
CRL	Certificate Revocation List
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
HN	Home Network
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IPsec	IP secure
IPv6	Internet Protocol version 6
JFK	Just Fast Keying
MAC	Message Authentication Code
MANA	MANual Authentication protocol
MD5	Message Digest #5
MN	Mobile Node
N_1	Nonce (Number 1)
OCSP	Online Certificate Status Protocol
PAN	Personal Area Network
PFS	Perfect Forward Secrecy
PKI	Public Key Infrastructure
RSA	Rivest – Shamir – Adleman
SA	Security Association
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SM	Security Module
SRP	Secure Remote Password protocol
SSL	Secure Socket Layer
TE	Terminal Equipment
TLS	Transport Layer Security
TRD	Tamper-Resistant Device
USIM	Universal Subscriber Identity Module
WAP	Wireless Application Protocol
WIM	WAP Identity Module
WTLS	Wireless Transport Layer Security

Executive summary

This document is the second of three documents in the work package WP4 of the EU-funded IST project SHAMAN and presents an intermediate specification of security modules (SM). It is based on the fundamental requirements listed in the earlier deliverable D05 and the work performed in work packages WP1, WP2 and WP3, concerning secure access to heterogeneous networks, secure communication inside a personal area network (PAN) and the realisation of a personal certification authority (personal CA). Its purpose is to identify the features a SM has to provide in order to support the requirements from those work packages, to define a reference model which serves as a basis for the implementation for a SM, and to make a statement about its feasibility using today's standards for tamper-resistant devices.

Security modules (a definition for the term has been given in D05) are secure containers and execution environments for secret data and algorithms operating on the secret data. However, their resources - storage space and processing power - are limited. Although these resources increase constantly with evolving technology, it is still a strong demand to use them efficiently. The challenge consists in finding the right distribution of functionality between the SM and the rest of the system which grants a reasonable performance but also a high level of system security. As a rule of thumb, long-lived secrets should be created inside the SM and never revealed to the outside world, whereas short-lived secrets could exist outside the SM (as long as compromising them does not enable other more serious attacks).

WP1 is studying network access security and therefore requires support from the SM for authentication and session key agreement, and the protection of the access link using the agreed session key. Authentication using shared key methods (the AAA for IPv6 protocol) as well as public key methods (IKE protocol) has been investigated. A functionality split between the terminal and the SM has been worked out for both; the required new SM operations are described and visualized in message flow diagrams, and the basic SM functions needed are listed. For the protection of the access link, it could turn out useful to store session keys and other security context information on the SM.

For WP2, the SM must support secure PAN internal communication, especially component initialisation, subsequent authentication, integrity and confidentiality protection. The SM's tasks are mainly the derivation and storage of key material; it remains to be clarified if policy information for secure download should also be stored on the SM. Additionally, it is pointed out how the workload for the SM can be reduced by distributing the cryptographic operations, e.g. in the case of server-aided secret RSA computation.

For WP3, the SM is especially interesting in the context of the personal CA. The client of the CA, as well the CA device itself, may require the help of a SM, namely for tasks such as key pair generation, secure storage and execution, certificate handling and revocation.

All these requirements for the SM have been summarised and grouped in the form of a reference model. Due to limited resources, it may be that not all of the desired functionality can be implemented at once on one SM; therefore, three different levels of security have been defined: "intermediate", "high" and "personal CA". The basic functions and protocol support capabilities for each of these levels have been listed in a concise form which could serve as the basis for implementation of a SM.

It has also been investigated how much of the reference model is already feasible on today's existing devices. One example is the WAP identity module (WIM): Although SHAMAN's focus is clearly outside of WAP, the WIM standard says that modules adhering to it can also be used for non-WAP applications. The interesting thing is that mature realisations of the WIM as a smartcard exist, which fulfil all basic requirements of the intermediate level of a SHAMAN SM and several of the high level requirements. However, the support for the AAA and IKE (or son-of-IKE protocols) is missing.

For the final report, further work is still required to study the functionality split of other public key protocols, certificate handling and the establishment of security associations. Additionally, a more

complete description of the software (like the minimum set of required algorithms) and hardware basis of the SM will be given.

1 Introduction

1.1 Purpose and background of this document

This document, the second of three deliverables in WP4 of the SHAMAN project, presents an intermediate specification for a security module (SM). It provides a first version of a reference model and a software- and hardware-specification of the SM. This will be further refined until it reaches a stable state in deliverable D13, which then serves as input for implementors and standard organisations as well as a stimulus for progress in smart card technology. It also provides input for the demonstrator to be developed in WP5, however not every aspect of the SM will be implemented in WP5, and it is not the only purpose of WP5 to demonstrate the SM.

This document is based on the deliverable D5, where the requirements for an SM have been identified, and on the interim results of work packages WP1-WP3.

1.2 Overview

After this introduction, the subject of a reference model for the SM is defined in section 2.

Section 3 is devoted to the functionality of security modules required by the other work packages (WP1–WP3).

In Section 4, a reference model is presented which is a synthesis of the principles developed in the previous section.

Section 5 introduces possible realisations of the reference model and existing products which could be used as a security module.

Conclusions for further work are drawn in Section 6.

2 Subject of a reference model

In the deliverable D05 [1], a list of requirements and challenges for a security module (SM) were identified. The objective of a reference model is to cast this input into an architecture which can be implemented and, in turn, be used as input for standardisation activities.

Traditionally, a security module like the SIM in GSM, was a smartcard which contained a number of keys, personal data (like the IMSI) and some algorithms (see the GSM security description in 2.1.5.1 of D02 [2]). In the future, the complexity is expected to increase; for example, a personal CA could be implemented or supported on a SM, or the performance of certain algorithms could be improved by distributing them between the SM itself and a trusted server (without compromising security).

A security module is a “tamper-resistant device”, as defined in D05, section 1.3. Tamper-resistant devices suffer from limited resources, and therefore it is often desirable to off-load certain functionality to other, more powerful devices. In this case, there is always a trade-off between security and efficiency. The question is how to judge the security of the link between the mobile node and the access point, between home network and visited network, between SM and the device it resides on (let’s call it SM-TE interface) and so forth. Therefore, certain protocol implementations will need to share the work between the SM and the TE or eventually other devices which act as a trusted server. Examples for this are given in section 3.

Elements of a reference model can be algorithms, data to be stored, file structure and process model of a smart card, interfaces, communication and trust relations between entities of one network or different networks as well as hardware requirements which arise from the desired functionality. Hardware constraints could lead to limitations on the model. In D05, it was mentioned that smart cards will remain to be limited devices, although their storage space and processing power continues to increase.

It remains to be defined which protocols the SM should support in the SHAMAN scenario. In WTLS, for example, the WIM (WAP identity module) acts as a service access point (SAP) for several layers in the protocol stack. For the sake of a proper design, the SM specification should strive for a well-defined interface and separation between the functionality of the SM and the mobile equipment that makes use of it; if the realisation of the mobile equipment changes slightly, there should be no necessity to make a redesign of the SM as well.

There may be several versions of the reference model - or several realisations of one reference model - depending on the desired degree of security or functionality. This is further explained in section 4.

References

[1] ‘Intermediate report on the role of security modules in heterogeneous networks, distributed terminals and PKI’, Deliverable D05, IST-2000-25350-SHAMAN

[2] ‘Intermediate Report: Results of Review, Requirements and Reference Architecture’, Deliverable D02, IST-2000-25350-SHAMAN

3 Required functionality

In this section, the functionality required for a SM is deduced from the scenarios developed in SHAMAN's work packages 1-3.

3.1 For WP1: Network access security

In WP1/D02 requirements for securing the access link between a mobile node and a heterogeneous network were presented. The mechanisms required to be supported by the mobile node (= terminal equipment + security module) can be split into two:

- authentication and session key agreement;
- protection of the access link using the agreed session key.

In D05 the requirements on network access security that relate to the SM were discussed. In this document we take this further and consider how the data and functionality required to support these mechanisms should be split between the terminal equipment and the security module (SM). Regarding authentication and session key agreement, mechanisms based on secret key and public key techniques are investigated separately.

3.1.1 Authentication using secret key techniques

AAA for IPv6 is a candidate authentication and key agreement protocol based on secret key techniques which is being investigated by WP1. The protocol is based on the authentication protocol from "draft-perkins-aaav6-05" [1] incorporating key distribution ideas from "draft-ietf-aaa-diameter-mobileip-09" [2]. Annex A contains message flows and a very straightforward functionality split for the protocol. Annex A also defines the protocol parameters which are referred to below.

The choice of algorithms F1, F2 and F3 depends on the security association shared between the security module and the home network. The HMAC_MD5 algorithm using Ka as the key is suggested to be used for F1 and F2 in [1], whereas a simple hash using the MD5 algorithm applied to Ka plus the other algorithm inputs is suggested to be used for F3 in [2].

In [1] replay protection is provided using either random challenges or timestamps. In this document we only consider the use of random challenges.

Message flows for two versions of the protocol are specified:

- Basic protocol
- Basic protocol with an additional Home Network (HN) challenge

If the additional HN challenge is unavailable at the start of the protocol run then the terminal equipment (TE) must request it from the HN first. This requires an extra round trip between TE and HN.

Regarding the functionality split, it is suggested that it is feasible to perform all authentication and key agreement functions at the mobile node side on a smart card based security module. This requires two commands to the card. There are two variants of these two commands depending on whether the extra HN challenge is used - the optional N_4 and N_4' parameters being included in square brackets.

- User authentication command:
  - input: N_1;
  - output: N_2, [N_4',] ID, AUTN1
- Network authentication and session key agreement command:
  - input: N_3, [N_4,] AUTN2;

- output: Ks

It should be noted that both versions of the protocol are stateful as far as the security module is concerned. In particular, in the basic and enhanced versions the security module is required to store the N_1 and N_2 parameters from the first command since these are used to verify AUTN2 send by the network when processing the second command. Furthermore, in the enhanced version of the protocol, the N_4' value must be stored from the previous run of the protocol.

3.1.2 Authentication using public key techniques

In this section we consider the IKE protocol as a candidate authentication and key agreement protocol for network access which is based on public key techniques.

3.1.2.1 Description of the protocol

The Internet Key Exchange is specified in Internet RFC 2409 [3]. It is a method to set-up Security Associations (SA). IKE is typically used to set-up SAs for IPsec but it may be used to set-up SAs for other security services.

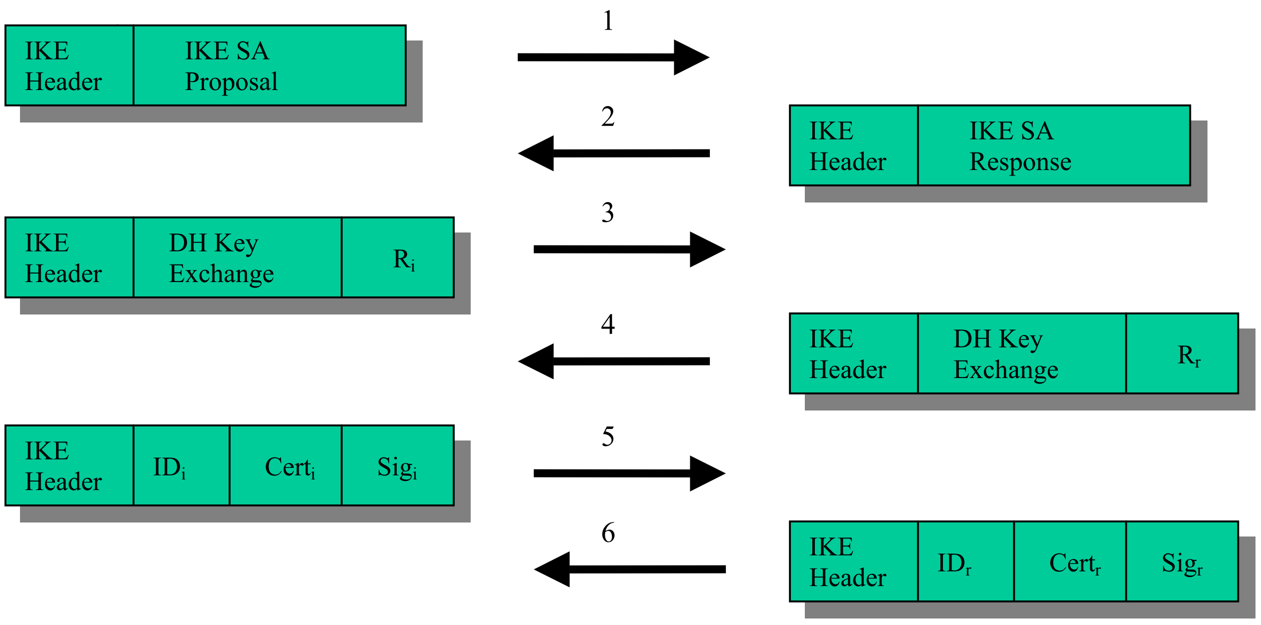
There are two phases in IKE. Phase 1 establishes an IKE SA while Phase 2 uses that SA to negotiate new SAs for another protocol, e.g. IPsec. Typically the IKE SA established in Phase 1 can live on to secure several subsequent Phase 2 exchanges.

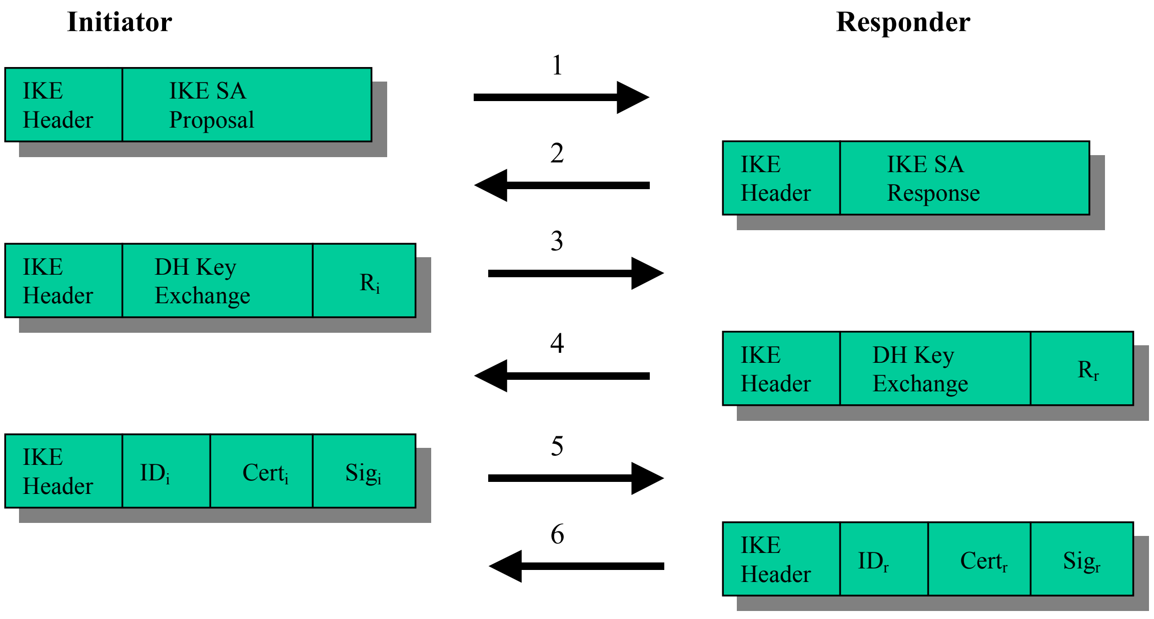
Phase 1 has two modes. Main mode is mandatory whilst aggressive mode is optional. The main mode exchange involves three round trips. Aggressive mode has one less round trip but does not have the same negotiation capability as main mode. This section only considers main mode.

Phase 1 uses Diffie-Hellman for key agreement and one of the following three methods for authentication:

- Signature using RSA or DSA;
- Public key encryption using RSA;
- Pre-shared secret.

The following shows the message flow where authentication is performed using signature or pre-shared secret. The deviations in the message flow when public key encryption is used for authentication are indicated.





The main mode consists of six messages that are exchanged between the initiator and the responder to establish an IKE SA.

The initiator sends an IKE SA proposal, listing all the supported authentication methods, Diffie-Hellman (DH) groups, choice of encryption and hash algorithms and desired lifetime of the IKE SA.

The responder sends an IKE SA response indicating its preferred authentication method, DH groups, encryption and hash algorithm and acceptable IKE SA lifetime.

Once the two parties have agreed a common set of methods, the protocol resumes with a DH key exchange protocol as follows. When public key encryption is used for authentication the random values sent in the DH exchange are encrypted with the recipient's public key.

The initiator sends his DH public key g^x mod n plus a random value R_I.

The responder sends his DH public key g^y mod n plus a random value R_r.

The protocol concludes with an authenticated exchange.

The initiator sends his identity ID_I and optionally a certificate linking the identity to the public key. This is followed by a hash over certain data including a secret string (SKEYID) which is determined based on the type of authentication method used.

The responder sends his identity ID_r and optionally a certificate linking the identity to the public key. This is followed by a hash over certain data including a secret string (SKEYID) which is determined based on the type of authentication method used.

At the end of the exchange, both parties can compute the following key material from SKEYID, the DH secret (g^{xy} mod n), and various other data:

- SKEYID_e: key material for encrypting Phase 2 exchanges;
- SKEYID_a: key material for authenticating Phase 2 exchanges;

SKEYID_d: keying material for protocol whose SA is negotiated during Phase 2.

3.1.2.2 Analysis of security critical data

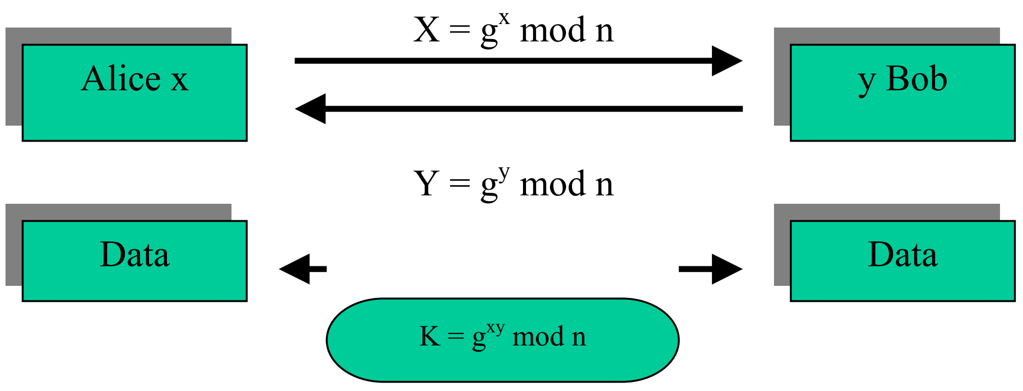
IKE uses the following critical security data at the mobile node:

- The DH private keys x and y
- Random values Ri, and Rr
- The DH public keys g^x mod n and g^y mod n
- The DH secret, g^{xy} mod n
- The authentication secret
- The secret string, SKEYID
- The key material for protecting Phase 2 exchanges, SKEYID_e and SKEYID_a
- The key material for the protocol whose SA is negotiated during Phase 2, SKEYID_d
- Root keys to verify certificates

In the following subsections we consider each of the critical data in turn to determine which security functions should be implemented in the security module and which (if any) can be implemented in the terminal equipment.

3.1.2.2.1 The DH private keys x and y

The DH private keys x and y are used in the DH key exchange. Both keys need to be kept secret because if one of them is obtained by an attacker he could generate the DH secret for himself using the corresponding public key which is passed in the clear in the IKE handshake. The diagram below shows the exchanges needed to generate a DH secret.



An attacker needs only to obtain either the value x or y to generate the shared DH secret. To confidentiality protect the DH private key on the security module it is necessary to implement the DH operations on the security module.

3.1.2.2.2 The random values Ri and Rr

The random values Ri and Rr are used to compute the SKEYID. If the random number generator is compromised then it could lead to attacks. It would be desirable to use a random number generator on the security module rather than have to trust an external source on the terminal.

3.1.2.2.3 The DH public keys

The DH public key need not be stored on the security module for security reasons but it may be more convenient to provision it on the security module rather than on the terminal.

3.1.2.2.4 The DH secret

If the DH secret is obtained by an attacker, then it could be used together with the SKEYID to determine the key material used for protecting Phase 2 exchanges and key material for the protocol whose SA is negotiated during Phase 2.

To confidentiality protect the DH secret on the security module it is necessary to implement the functions for deriving the various other key material on the security module.

3.1.2.2.5 The authentication secret

Different secrets are used for authentication depending on which method is used:

- Signature: RSA private signature key or DSA private signature key
- Public key encryption: RSA private key
- Pre-shared secret: The pre-shared key

For authentication with signatures a hash on the secret string SKEYID and various other data is signed and verified. For authentication with either public key encryption or pre-shared keys, the hash on SKEYID directly authenticates the exchange because SKEYID is generated from information which can only be obtained by the holder of the RSA private key or the pre-shared key.

If the authentication secret is revealed then it could allow an attacker to masquerade as a legitimate user. To protect the authentication secret it is necessary to implement computations based on that secret in the SM, and to make sure that the secret never revealed outside the SM.

3.1.2.2.6 The secret string, SKEYID

If the secret string SKEYID is obtained by an attacker, then it could be used together with the DH secret to determine the key material used for protecting Phase 2 exchanges and key material for the protocol whose SA is negotiated during Phase 2.

To confidentiality protect the SKEYID on the security module it is necessary to implement the functions for deriving the various other key material on the security module, and to make sure that the SKEYID never revealed outside the security module.

3.1.2.2.7 The key material for protecting Phase 2 exchanges, SKEYID_e and SKEYID_a

If SKEYID_e and SKEYID_a are obtained by an attacker, then they could be used to eavesdrop, modify or spoof messages in a Phase 2 exchange.

To confidentiality protect the SKEYID_e and SKEYID_a on the security module it is necessary to implement the protection of the IKE Phase 2 exchanges on the security module, and to make sure that SKEYID_e and SKEYID_a are never revealed outside the security module.

3.1.2.2.8 The key material for protocol whose SA is negotiated during Phase 2, SKEYID_d

If SKEYID_d is obtained by an attacker, then it could be used to generate the key material established during the Phase 2 exchange. (If Perfect Forward Secrecy (PFS) is provided then the attacker would also need the DH secret generated during the Phase 2 exchange.)

To confidentiality protect the SKEYID_d on the security module it is necessary to implement the algorithm for generating the key material on the security module, and to make sure that SKEYID_d is never revealed outside the security module.

3.1.2.2.9 Root keys to verify certificates

If public key signature or public key encryption is used for authentication then IKE allows both parties to exchange certificates which may be used to obtain trusted copies of the necessary public keys. In order to be able to verify these certificates, root public keys need to be provisioned in the mobile node. The root keys need not be stored on the security module for security reasons but it may be more convenient to provision them on the security module rather than on the terminal.

3.1.2.3 Functionality split between terminal and security module

Annex B considers the following two functionality split scenarios:

1. The security module is only involved in mutual authentication so that after Phase 1 the terminal can engage in Phase 2 exchanges to create new SAs for other protocols without the security module.
2. The SKEYID is never revealed outside the security module so that the terminal cannot engage in Phase 2 exchanges to create new SAs for other protocols without the security module.

If Phase 2 exchanges are commonly used to "refresh" session keys, then it is possible that the mobile node may engage in secure communications with the network for long periods of time without the presence of the security module being verified by the network. This may be a security vulnerability due to threats associated with 'untrusted' terminals. To avoid such threats, the network could force the use of a Phase 1 exchange to verify the presence of the security module when the session key needs to be refreshed. However, this is not as efficient as a Phase 2 exchange. It is therefore preferable to be able to verify security module presence during Phase 2 exchanges.

3.1.3 Protection of the access link

Protection of the access link generally involves authenticating and encrypting the potentially large amounts of user traffic and signalling information that is sent on the access link. As a consequence, it is generally considered neither cost effective nor feasible to implement this bulk protection on a smart card based security module because of the limited processing and storage constraints, and the limited bandwidth on the card interface. However, there may be other requirements on the security module to support access link security. For example, it may be desirable to store session keys and security context data used to protect the access link on the security module. These issues are discussed in the following sub-sections.

3.1.3.1 Storage of access link session keys on the security module

In mobile systems it may be desirable to be able to establish a new connection without having to re-authenticate with the network to establish a new session key. This is because the execution of an authentication protocol may add a significant delay to the connection establishment procedure or it may impose a significant load on the signalling network and authentication server especially when connections are short-lived.

To allow a security session to be resumed without running a new authentication and key agreement protocol, the session key must be stored in either the terminal equipment or in the security module at the end of each connection. However, to allow for so-called "plastic roaming" where a typically smart card based security module is moved between terminals, it is an advantage to store the session key in the security module since the user can resume a secure session on the new terminal without having to re-authenticate. If the session key were stored in the terminal, then the terminal would clearly not be able to use the stored key if a new security module was inserted. Note that plastic roaming is still possible even if the session key is not stored in the security module since the terminal can simply execute the authentication and key agreement protocol to obtain a new session key when a new security module is inserted.

A further advantage of storing the session key on the security module is that it can provide better protection for the key when it is not in use. This may be important if the session key is not used for a long period of time (e.g. if the user is inactive for long periods between connections). Although the key must be exposed to the terminal when it is being used, it may be an advantage to limit that exposure by ensuring that it is securely stored in the security module (and securely deleted in the terminal) when it is not being used.

3.1.3.2 Storage of other security context information on the security module

As well as session keys, the terminal (and/or the security module) may be required to store other information in order to allow a secure session to be resumed. This may include information that was negotiated during the SA establishment with the network when the session keys themselves were generated. In the case of IKE, the security context information would include algorithm information, SA lifetime, etc.

If it is required to be able to resume a session without re-authenticating when the security module is moved between terminals (plastic roaming), then it may be required to store all security context information on the module. However, depending on the amount of security context data, it may not be practical or cost effective to store all this information on the security module because of memory limitations and interface bandwidth constraints. As an alternative, it may be possible to be able to renegotiate the security context information without requiring a full authentication and key agreement.

This depends on whether the security association set-up procedure can be decoupled from authentication and key agreement.

Storing other security context information on the terminal rather than on the security module also has the effect that it is potentially easier for an attacker to obtain, modify or delete certain security context information. However, threats against this sort of information are much less severe than threats against session keys.

3.1.4 General principles

The following general principles are proposed to be used to determine the security functionality split between the security module and terminal for a particular security protocol:

- All long-term secret/private keys required at the user-side shall be stored and used within the security module such that their values are not revealed outside the security module. Long-term keys are defined as keys with a lifetime of more than 24 hours. Examples of long-term keys include authentication keys that are issued to the customer when the subscription is created and are then used throughout the duration of the subscription.
- Some short-term secret/private keys may be used outside the security module if it is not feasible or cost effective to perform the cryptographic computations inside the security module. If such keys need to be stored for long periods when they are not being used on the terminal, then it is preferable to store them on the security module. Short-term keys are defined as keys with a lifetime of less than 24 hours. Examples of short-term keys include session keys used to protect the confidentiality and integrity of the access link.
- Subject to the second principle above, security context data (which will typically include short-term secret keys) may be stored on the security module if plastic roaming is required to be supported. Note however that plastic roaming can still be supported even if security context data is not stored on the security module.
- Any random numbers required by the security functions implemented on the security module shall be generated on the security module for security reasons to avoid having to trust random numbers provided by the terminal. If the terminal also requires random numbers then it may use the random number generator on the security module to avoid duplication of functionality. However, any random numbers required by the terminal may also be generated within the terminal itself – there is no security advantage in using random numbers generated in the security module assuming that the quality of random numbers generated by the terminal is at least as good as those generated by the security module.

3.2 For WP2: Distributed terminals

In WP2/D03 security requirements for distributed terminals have been presented, some of these requirements are SM related and those are listed in document D05. SM related requirements in D05 are divided into two classes whether they are relevant for PAN internal or external communication (or both). In this section we are focusing on PAN internal communications and secure download and our goal is to show how SMs can be implemented that fulfil these requirements.

3.2.1 Secure PAN internal communication

Secure PAN internal communication means that communication between trusted a PAN component is encrypted, authenticated and integrity protected. Every pair of trusted components must have the ability to do mutual authentication. Two types of trusted components can exist in a PAN depending on whether the component is a first party component or a second party component. With respect to a reference component (i.e. a component we start looking at the different other components within the PAN), all components that can be authenticated an which belong to the same owner as the reference component are considered first party components. Likewise, all components that can be authenticated but which belong to a different owner than the reference component are considered second party components. The remaining components, i.e. those that can't authenticate or that are not (yet) authenticated belong to the class of untrusted components. Furthermore it was required that every first party component must have the ability to distinguish a first party component from a second party component. Two first party components **must** have the ability to make secure and authenticated key exchange without any user interaction.

Both symmetric and asymmetric cryptographic techniques use secret/private keys for authentication. If secret keys are stored in a PAN component, then they should be stored on tamper-resistant memory, which can be considered to be part of the SM. Note that PAN components do not necessarily need to store secret data, for example, components can base their authentication procedure on passkeys that can be human memorable.

Our goal in this section is to present what are the needed resources in SHAMAN compliant SMs for protecting PAN internal communication. SM resources depend on key management techniques, PAN scenarios from WP2 and the used cryptographic algorithms.

3.2.1.1 SM requirements on component initialisation

The wireless link between PAN components is inherently insecure, therefore first party components must have a link that has been secured either physically or cryptographically. Physical security is possible for example by using fixed connection, human involvement, low power channel, e.g. Physical techniques are not very flexible and often requires active user involvement. A better solution is to put cryptographic information in components to personalise them and make authentication possible - this procedure is called imprinting.

WP2 currently provides two imprinting mechanisms; one uses a manual authentication protocol (MANA) and it is used for imprinting a symmetric system the other one uses an imprinting mechanism for an asymmetric (public key) system.

In the first imprinting mechanism parties exchange Diffie-Hellman values g^x and g^y, where g is generator of group that is used in the Diffie-Hellman key-exchange and x and y are randomly chosen integers. The device then executes the MANA protocol for data $D=(g^x, g^y, \text{text})$ where text is any additional data e.g., each other's identities, that components may want to agree upon. MANA will provide that the devices are mutually authenticated and that they have the shared key $K=g^{xy}$. The required resources for the SM are

- Pseudorandom generator (recommended).
- Computation of Diffie-Hellman exponentiation (recommended).
- Derivation of initial secret from shared Diffie-Hellman secret (recommended).
- Computation of one-way function (mandatory)
- Storage of key (mandatory)

It is recommended that no sensitive information about the key be leaked out of SM at any time. Therefore it is recommended that SMs inside the components choose integer x, y and compute Diffie-Hellman values g^x and g^y. A one-way function is needed for deriving further key material.

In the second imprinting mechanism the PAN component will get a private key and a public key that has been certified. Two approaches exist for key generation; the private key is either generated in the component or it is generated externally. For the SM there is a possibility that the private key is generated in the component, but not in the SM. From the SMs point of view that case is just like any other case where the private key is generated externally.

Needed resources for SM are

- Private key generation (if private key is generated in the SM) (mandatory)
- Pseudorandom generator (mandatory)
- Public operations (optional)

- Private key operations (mandatory)

A pseudorandom generator is needed for creating a private key.

3.2.1.2 SM requirements on subsequent authentication

As a result from initial authentication, the respective parties will have strong authentication keys. According to current proposals in WP2 these keys will also be used in subsequent authentication unless the keys have expired or are not usable for other reasons. Therefore the authentication algorithm must reside on the SM.

3.2.1.3 SM requirements on integrity protection

According to the document D03 in PAN both data and signalling channels should be integrity protected. As currently proposed in WP2 integrity protection should rely on the usage of MACs instead of only encryption. From an SM point of view there are now two possibilities:

- The integrity key will be derived in SM (mandatory)

In this case the SM should have the ability to compute an integrity key as an output of a one-way function from the secret that has been installed on the SM. After derivation the SM can provide the integrity key to the PAN component, that can then compute the MACs.

- MACs will be computed in SM (mandatory)

The SM should have the capability to compute MACs. This is essentially the same requirement that the SM should be capable to compute values of a one-way function. However this method is inefficient because it will require that huge amounts of data are sent to and from the SM.

3.2.1.4 SM requirements on confidentially protection

According the document D03 in PAN both data and signalling channels should be confidentially protected. As in integrity protection two possibilities for implementation of SM exist:

- The confidentially key will be derived in SM (mandatory)

In this case the SM should have the ability to compute a confidentially key as an output of a one-way function from the secret that has been installed on the SM. After derivation the SM can provide the confidentially key to the PAN component, which can then encrypt/decrypt communication data.

- Encryption/decryption is done in SM (mandatory)

This approach will require great computation power and fast I/O capabilities for the SM. This kind of solution may not be possible if the SM is implemented according to smartcard specifications like ISO-7810 [10]. However, smartcard technology can provide faster I/O capabilities for example, using the communication protocol T=1 (asynchronous half-duplex block transmission). Instead of serial interface also PCMCIA and USB interfaces can provide desired I/O capabilities.

3.2.1.5 SM requirements on identification and location privacy

When a connection between trusted PAN components is established then these components will mutually authenticate each other. This authentication procedure should not reveal a PAN components identity to outsiders. In WP2 a challenge response based authentication protocol is being investigated. Firstly, the service provider component SP sends a random challenge RAND to the service requestor component, which then replies with the response RES to the SP. This RES is calculated from the RAND by using a one-way function and the shared-secret, that has come from the imprinting process or is further derived from it.

The SM can derive the RES from the shared secret and RAND, or the RES can be derived by an entity outside the PAN if the key that has been derived from initial-secret from imprinting is available to that entity. In both cases a one-way function in SM is needed.

3.2.2 Secure download

Components in a PAN may have different resources and therefore it is desired that the components can share these resources. Assume a case where none of the PAN components have a required functionality. In this situation one component can download an executable, (which provides the functionality) and then further distributes it to other components. Resource sharing and downloading executables from outside the PAN requires policies, which tells what are the resources that a particular component can have access to.

Policies and access control mechanisms do not make any additional requirements for the SM, because if policies require that communication should be secured, the methods in sections 3.1, 3.2.1 and 3.2.2 can be used. An additional requirement on the SM comes from the fact that origin and integrity of the message must be verified. SMs may contain or get symmetric secret keys for origin and integrity verification and the SM should do these verifications itself or derive further key material using a one-way function which is passed to the component to allow it to do the verifications instead. If public key techniques are used for verification, the SM may contain the public key of the root CA, because it can then verify the downloaded certificate, which should be provided along with the downloaded content. By using this certificate the SM may verify the signed digest of the content. Requirements for SM are:

- Storage of root CA's public key (optional)
- Computing digest of the content (optional)
- Verifying signature of the digests of the content (optional)

The SM can verify the integrity of the downloaded content if the digest of it is computed in the SM. However this consumes a lot of the SM's resources.

3.2.3 Conclusions

It is required that all first party components have the ability to authenticate each other and make a secure key exchange without any user action. This goal can be achieved by imprinting these components. The result of imprinting determines the shared initial secret that must be stored on the SM. It is possible that components can be monitored at the time when this initial secret between the two components is generated if carried out on a non-tamperproof device, therefore it is recommended that secrets be generated in the SM. The challenge-response algorithm should be protected against anti-replay attacks. The same key must not be used every time, therefore the SM should provide a mechanism for deriving a new key for the challenge-response algorithm.

Once the components are imprinted, the shared initial secret is used to secure the communication between the components. It is required that the SM contains the authentication algorithm. The one-way function is needed for deriving the key material for the encryption and integrity algorithms and the challenge response algorithm for identification and location privacy.

Secure download requires that the origin and the integrity of downloaded content must be verified. The SM may verify the certificate and the signature of the digest of this content. If the integrity of downloaded content is verified in the SM, then the digest of content must also be computed in the SM.

3.3 For WP3: Personal PKI

We consider the security module requirements associated with the use of a 'Personal PKI'. We divide this discussion into three main parts:

- *Mobile device requirements*, i.e. the PKI requirements for a security module used with a 'client' mobile device (i.e. one which is a client of the Personal CA);
- *Personal CA requirement*s, i.e. the PKI requirements for the security module used by the personal CA device.
- *General requirements*, i.e. the PKI requirements applying to both Personal CA and client device.

3.3.1 Mobile device (client) requirements

The PKI-support requirements for a 'client' mobile device (i.e. one which is a client of the Personal CA) are as follows.

1. Key pair generation, i.e. the secure generation of one or more asymmetric key pairs for the mobile device;
2. Private key secure storage, i.e. the provision of facilities to store the mobile device private key(s) in a way which preserves its(their) confidentiality and integrity;
3. Root public key storage, i.e. the provision of facilities to store the public key(s) of the Personal CA(s) in a way which preserves their integrity;
4. Secure computation with device private key, i.e. the performance of cryptographic computations using the mobile device private key in a way that does not threaten its confidentiality or integrity;
5. Public key revocation handling (e.g. verifying CRLs or OCSP status messages supplied by the Personal CA).

It would certainly be highly desirable for requirements 1, 2 and 4 to be met by the security module for the mobile device. Meeting requirement 3 would also be desirable, although there exists the possibility that this and other information requiring integrity protection could be stored externally to the security module and integrity protected using a MAC computed using a key stored internally to the module. Requirement 5 is probably not something that needs to be handled within the SM.

3.3.2 Personal CA requirements

The PKI requirements for the Personal CA device are as follows.

1. Key pair generation, i.e. the secure generation of one or more asymmetric key pairs for the Personal CA;
2. Private key secure storage, i.e. the provision of facilities to store the Personal CA private key(s) in a way which preserves its(their) confidentiality and integrity;
3. Secure computation with Personal CA private key, i.e. the creation of public key certificates and, possibly, CRLs, using the Personal CA private key in a way that does not threaten its confidentiality or integrity.

The support for all three of these functions by a Personal CA security module would appear to be a high priority. Other possible functions for a Personal CA security module include the following:

1. Support for secure auditing, i.e. the support of integrity-protected records of all security-critical events (e.g. key generation, certificate creation, etc.);
2. Support for signed certificate status management messages, e.g. CRLs, or as required by OCSP.

3.3.3 General requirements

The exact set of cryptographic functions which need to be performed within the module as opposed to externally is negotiable. For example, in some circumstances it may be acceptable to perform part of the message processing for a signature computation externally to the secure device – how acceptable this is depends very much on the design of the cryptographic primitive. For example, the PSS-R based signature scheme in ISO/IEC 9796-2 [11] has been designed to permit some processing externally to a secure device without endangering the private key – this is not the case for all signature schemes.

All PKI-supporting security modules will need to be equipped with an authorisation function, to prevent unauthorised generation of new key pairs (and associated erasure of old key pairs).

3.4 Distribution of security

In D05 section 6.3, it was explained that SM might have limited resources. This resource problem can be solved in some cases by using distributed security. Our goal in this section is to present how SM or security services of SM can be distributed.

3.4.1 Distributed SM model

In D05 it was explained that it might be possible that a group of SMs could be connected to each other. Individual SMs can be considered to be security "atoms" and a group of SMs form a security "molecule". In the distributed SM model similar scenarios like in the case of a PAN arise, i.e., the centralised scenario where all SMs (atoms) are connected to a central SM or a fully-meshed scenario where all SMs are connected to each other. Different kinds of SMs with different resources/properties could be specified. However, for simplicity we only consider the case where all SMs have the same resources/properties.

In the distributed SM model task management is an important issue. The first possibility is that SMs gets their orders on what they should do from a PAN component and this PAN component must be aware of what all other SMs are doing. This first possibility will require the PAN component that is task manager to have some specific role in the PAN, for example it is the PAN manager component. The second possibility is that the task management is done by SMs only. Because all SMs have the same resources, they all must be capable to do task management. A different question is whether all SMs should participate in task management.

Security of the distributed SM model is analogous with security of PAN internal communications. Both data and signalling (related to task management) between SMs should be authenticated, confidential, integrity protected and perhaps anti-replay protected.

The distributed SM model can provide scenarios where some SMs can generate keys to other SMs and then distribute these keys in a secure way to these other SMs

3.4.2 Distributing security functions between SM and MN

In D05/WP4 it was explained how security functions should be distributed between a SM and a device that is not secure. One example of a security function that can be distributed is the IKE protocol, see 7.2 (Annex B). In document [5], it has been noticed that IKE is a very complex protocol and has therefore a lot of problems. For example, most IKE implementations are such that the protocol will not only reject unrecognised payloads, but will also force a termination of the connection. A connection can be terminated when optional payloads that are not even security-critical are not recognised. IETF IPSec working group has now proposed a new algorithm IKEv2 [6]. Two other proposals, Just Fast Keying (JFK) and SIGMA have also been proposed. However IKE is the current standard and it provides key exchange in many different situations, for example IKE can provide SAs for security protocols like IPSEC and MAPSEC. These can be used to protect PAN external and perhaps even PAN internal communication.

A public key operation requires a lot of resources; therefore in some situations it may be good that public key cryptographic primitives like RSA encrypt/sign operations are distributed. Many of the big CAs such as Verisign, Baltimore and Entrust provide certificates for RSA public keys and actually RSA is the only public key cryptographic system that is supported by these CAs. So RSA can be considered as the standard in public key cryptography. In the next section we present a method for distributing RSA operations between a tamper resistant device (TRD) and an insecure device (non-tamper resistant device).

Server-Aided Secret RSA computation

A problem in server-aided secret RSA computation is that TRD computes $S=M^d \pmod n$, by using the help of a server. Here SK=<n,d,e,p,q> is the secret key such that n=pq is the RSA modulus, d is the secret exponent and p and q are the prime numbers. The TRD will use the help of a server (in our case the terminal equipment or PAN component), such that the server does not get any information that can help it to compute the value k^d (mod n) without the TRD. This means that the TRD does not leak any sensitive information out.

Several approaches in server-aided RSA computation exist [7,8]. The approach that has been presented in [7] is vulnerable to active attacks and the protocol in [8] is vulnerable to probabilistic active attacks. Presented here is a method from [9] that is secure against all known attacks.

The same terminology is used as in [9]. By a client we mean the TRD and by server we mean the terminal equipment or PAN component. First the client makes the following pre-computations (these are done only once after the RSA keys are generated):

1. The client chooses two random integers R and R', where $\lfloor \log R \rfloor < b$ and $\lfloor \log R' \rfloor < b'$, here b and b' are the given security parameters. Then it computes an addition-chain $\{a_R\}$ for R, this is a sequence $a_0, a_1, \ldots, a_l$, where $a_0=1$, $a_l=R$, $a_i=a_j+a_k$, when $0 \le j \le k < i \le l$.
2. The client randomly chooses r_i:s amongst the elements of the addition-chain for R, and r_i':s amongst segments of R', where $1 \le i \le k$ satisfy the following:

   $$R'=k_1 \| k_2 \| \ldots \| k_l,$$

   $$r_i'=k_i \times 2+1.$$

   Here notation || means concatenation.
3. The client computes

   $$t' \equiv 1/r_k(\ldots(1/r_1(d-r_1)-r_2)-\ldots r_k)-R \bmod \lambda(n),$$

   here $\lambda(n)$ is least common multiple of p-1 and q-1. The client prepares $u \equiv \prod(1/r_l) \bmod \lambda(n)$.
4. The client computes $w_p=q(q^{-1} \bmod p) \bmod n$ and $w_q=p(p^{-1} \bmod q) \bmod n$. Note that Chinese remainder theorem (CRT) implies

   $$M^a \times w_p + M^b \times w_q \equiv M^d \bmod n,$$

   where $a \equiv d \bmod (p-1)$ and $b \equiv d \bmod (q-1)$.

Next we present how the value M^d is computed in a secure way.

Client Server

Randomly choose d_1.
Compute $t=t'-u\times d_2 \bmod \lambda(n)$,
where $d_2=d- d_1$.

M,n, and $t \rightarrow$

Compute Z:
$Z=M^t \bmod n$.

$\leftarrow Z$

Compute z_p and z_q:
$z_p=(\ldots((Z \bmod p \times M_p^R)^{rk'}\times M_p^{rk})^{r(k-1)'}\ldots)^{r1'}\times M_p^{r1} \bmod p$
$z_q=(\ldots((Z \bmod q \times M_q^R)^{rk'}\times M_q^{rk})^{r(k-1)'}\ldots)^{r1'}\times M_q^{r1} \bmod q$,
here $M_p=M \bmod p$
Compute σ_p and σ_q:
$\sigma_p=d_2 \bmod (p-1)+\rho_p(p-1)$,
$\sigma_q=d_2 \bmod (q-1)+\rho_q(q-1)$,
where $0\leq\rho_p<p-1$ and
$0\leq\rho_q<q-1$ are randomly
chosen.

$\sigma_p,\sigma_q\rightarrow$

Compute y_p and y_q:
$y_p=M^{\sigma p} \bmod n$
$y_q=M^{\sigma q} \bmod n$

$\leftarrow y_p , y_q$

Compute $S=w_pS_p+ w_qS_q \bmod n$,
Where $S_p= y_p \bmod p$ and
$S_q= y_q \bmod q$.
Check if $S^e=M \bmod n$.

The performance of this protocol depends on the security parameters b, b' and k. The number of modular multiplications are 3/8(b)+3/4(b')+7/8(k)+11/2. In [7] two exhaustive search attacks have been presented. The better attack has a search space of size $2^{b+2b'+1}$, in the document [11] the recommended size should be 2^{64} and therefore the security parameters could be for example b=4, b'=30, k=5. However current requirements for size of search space is 2^{128}, this can achieved by choosing b=8, b'=60, k=5. With these parameters the number of modular multiplications is 58. In the case of a 1024-bit public key if M^d is calculated directly it will require approximately 1024 multiplications/squaring. The performance is approximately 20% better when the sliding window technique is used, however performances with sliding window techniques require much more RAM usage from the SM.

A very rough estimate with the protocol above is that the actual computation without pre-computations takes about 7% time compared to the usual method. However the server will do most of the work with this method and four messages need to transfer between the SM and the server.

3.5 Conclusions

The SM has limited resources; therefore it may have to distribute the workload. Two approaches have been presented, the first one is to distribute the SM and second one is to distribute security functionality between the SM and terminal.

The distributed SM model has following advantages

- Parallel computing between SMs is possible
- Different SMs may share secret data, like secret keys.

However the distributed SM has following disadvantages

- High I/O capabilities are needed
- Task management between SMs is problematic
- Connections between SMs must be secured, and this consumes SMs resources.

It has been shown how to distribute securely the IKE protocol between a SM and a terminal. It is identified that a single private key operation can be a complex task for a SM. A mechanism for distributing security for RSA private key operations has been presented. In case of 1024-bit RSA keys, the workload of computation for a SM is only 7% compared to standard RSA. However, using the distributed RSA method requires four messages to be transferred between the SM and terminal.

The distribution of RSA does not prohibit plastic roaming, although some pre-computed operations have to be carried out in this case.

3.6 References

[1] P. Flykt, C. E. Perkins, T. Eklund, AAA for IPv6 Network Access, Internet Draft, draft-perkins-aaav6-05, March 2002.

[2] P. R. Calhoun, T. Johansson, C. E. Perkins, Diameter Mobile IPv4 Application, Internet Draft, draft-ietf-aaa-diameter-mobileip-09, March 2002.

[3] D. Harkins, D. Carrel, The Internet Key Exchange (IKE), Internet RFC 2409.

[4] Nokia, Distributing Internet Key Exchange protocol between tamper resistant device and mobile terminal (Annex B)

[5] " Protocol Requirements for Son-of-IKE ", Internet draft, draft-ietf-ipsec-son-of-ike-protocol-reqts-00.txt

[6] "Proposal for the IKEv2 Protocol", Internet draft, draft-ietf-ipsec-ikev2-01.txt

[7] T.Matsumoto, K.Kato, and H.Imai, "Speeding up secret computations with insecure auxiliary devices", in Crypto'88, pp.497-506, 1988

[8] P.Beguin and J.J.Quisquater, "Fast server-aided RSA signatures secure against active attacks," in Crypto'95, pp. 57-69, 1995

[9] S.Hong, J.Shin, H.Lee-Kwang and H.Yoon, "A new Approach to server-aided secret computation", in ICJSC'98, pp. 33-45, 1998

[10] ISO/IEC-7810 , Identification cards, physical characteristics

[11] ISO 9796-2, Digital signatures giving message recovery, Part 2: Mechanisms using a hash function

4 Reference model

This section serves as a summary and reference and can be used as a basis for an implementation of a SHAMAN-compliant security module.

Several possible versions of the SM are introduced, qualified by their "security level", providing different basic functions and protocol support; these notions are defined in the following section.

In section 4.2 the required functions for each of the security levels are presented.

Section 4.3 gives some preliminary ideas about the SM's realisation.

4.1 Notions used in the reference model

4.1.1 Security levels

The importance of the SM features which have been worked out in section 3 differ from each other and depend on the desired level of security.

We define 3 levels here:

1. Intermediate
2. High
3. Personal-CA-enabled

The basic functionality which should be implemented on them is given in section 4.2.

"Intermediate" is the lowest security level (the term "low security" is not used here!). It supports only the functionality which is required as minimum. Whether this level is sufficient or not is always debatable.

"High" refers to the highest level of security. It fully includes the intermediate level functions, and provides some additional functions for higher security, but also convenience.

"Personal-CA-enabled" denotes a SM which includes all the functionality that is already included in the "High"-level and – additionally – enables the SM to be used in a personal PKI (for a client of a personal CA and for the CA device itself). The reason that this level was introduced is that some of the features for personal CAs require the SM to have considerable performance and storage capacity which may not be available with today's devices.

4.1.2 Protocol support

A SHAMAN-compliant SM must support the distributed version of the AAA [1],[2] and the IKE [3] (or "son-of-IKE") protocol.

An SM like a smartcard usually provides a defined command set. Each command reads a number of input parameters and returns a value which corresponds to a basic function like "hash", "digital signature" etc. In order to implement the mentioned protocols, this will be, in general, not enough: Some state information may have to be maintained or other (eventually more complex) commands must be added to the SM's command set.

4.2 Functionality for the different levels

In this section, the required basic and protocol functions for all 3 security levels (intermediate, high and personal-CA) are listed.

4.2.1 Intermediate level

The following basic functions are required:

- Pseudo-random number generator
- Storage of long-lived keys
- Private key operations (signature, decrypting)
- One-way function (for derivation of integrity or confidentiality keys, usage in the AAA protocol etc.)

Concerning the SHAMAN protocols: It makes sense to demand several different versions of the distributed protocols or their realisation from a high-level SM, but only the most basic ones from the intermediate-level, as follows:

- AAA protocol for IPv6, basic protocol (see 7.1)
- Distributed IKE, authentication with pre-shared key, simpler scenario (see 7.2)

4.2.2 High level

Additionally to the intermediate level, the following functions shall be provided:

- Storage of short-term keys (session keys, e.g.)
- Computation of Diffie-Hellman secret / derivation of initial secret from DH secret
- On-card generation of a key pair
- Storage of security context data (for plastic roaming)

Protocol support (additionally to those for the intermediate level):

- AAA protocol with additional HN nonce
- Distributed IKE, authentication with pre-shared key, complicated scenario
- Distributed IKE, authentication with signatures (either scenario)

Possibly, it is necessary to also include the validation of CRLs or OCSP protocol in the list, as for the personal-CA-level (see following paragraph). This is for further study.

4.2.3 Personal-CA-level

The following additional functions are needed:

- Storage of certificates
- Creation of certificates
- Validation of CRLs or OCSP protocol
- Secure auditing

The personal-CA-level does not provide any SHAMAN protocol support beyond those given for the high level (except perhaps the OCSP [5] protocol, but this is for further study).

4.3 Remarks about realisation

The algorithms used to realise the above mentioned basic functions are not specified in this document, and neither is the storage format of the data. This is left for further study. However, the client of the SM should be able to query the capabilities of the SM. It may also be necessary for the SM to support negotiation of capabilities as defined in certain protocols.

Probably, as already mentioned above, the command set of a common SM will not suffice to support the distributed protocols. For prototype versions of the SHAMAN-SM, this means that proprietary commands have to be introduced. Perhaps it will be possible in the future to extend the corresponding standards accordingly.

Unless the chosen protocols are stateless, state information must be maintained by any of the entities involved. Here the question arises if the SM must do this; the decision may depend on security considerations like the degree of trust between the SM and the TE, for example. A state machine might be rather costly to implement on a SM. Another possibility could be the use of cookies, as in IKE; they can be stored in a certain file on a smartcard.

4.4 References

[1] P. Flykt, C. E. Perkins, T. Eklund, AAA for IPv6 Network Access, Internet Draft, draft-perkins-aaav6-05, March 2002.

[2] P. R. Calhoun, T. Johansson, C. E. Perkins, Diameter Mobile IPv4 Application, Internet Draft, draft-ietf-aaa-diameter-mobileip-09, March 2002.

[3] D. Harkins, D. Carrel, The Internet Key Exchange (IKE), Internet RFC 2409.

[4] W.Aiello, S.M.Bellovin, M.Blaze, R.Canetti, Just Fast Keying (JFK), draft-ietf-ipsec-jfk-03, March 2002

[5] M.Myers et al., Online Certificate Status Protocol – OCSP,

draft-ietf-pkix-rfc2560bis-01, February 2002

5 Towards realisation: The WAP identity module

The role of smartcards as secure modules has been described in D05 of WP4. For GSM, the subscriber identity module (SIM) [1] already takes the role of a secure container and algorithm execution environment for personal data and keys. With increasing demands in telecommunication, new standards were developed like SIMAlliance [2], WIM [3] or USIM [4].

In this section, the WIM (WAP identity module) standard is described. A WIM card, possibly with some extensions, could be a promising candidate for the realisation of a SHAMAN-compliant security module.

5.1 Description

5.1.1 Purpose of a WIM

The WIM standard was specified by the WAP forum. WAP security functionality encompasses transport layer security (WTLS) [5] as well as application level security (supported by the WMLScript Crypto Library, for example).

Parts of the security functionality need to be performed by a tamper-resistant device, the WIM. Its principal purpose is to enhance security of the implementation of the "security layer" (the layer defined in the WAP architecture which hosts the WTLS protocol) and certain functions of the application layer. The WIM offers a service access point (SAP) which can be used by different layers of the WAP protocol stack. The information structure is based on the PKCS 15 [6] standard.

The WIM standard also says that "Use of generic cryptographic features with standard interfaces like ISO7816 and PKCS#15 can make it interesting to use the WIM also for non-WAP applications, like SSL, TLS, S/MIME etc.".

In order to find out if the WIM can also be used for SHAMAN's purposes, the following paragraph will have a closer look at the functionality defined in the WIM standard.

5.1.2 WIM functionality

A complete table of mandatory and optional functions of the WIM can be found in the WIM standard, chapter 14 (WIM static conformance requirements).

Roughly, the following functions are mandatory:

- Storage of key pairs
- Private key encryption (signing of hashes; unwrapping of keys is optional)
- Storage of certificates or URLs to certificates (for portability, mainly)
- Random number generation – based on hash function SHA-1
- WTLS key exchange
- Derivation of master-secret

Following functions are optional:

- Digital signature verification
- Elliptic curves (Diffie-Hellman, DSA)

The most striking thing about this list is that it does not include any symmetric encryption algorithms. This makes the WIM complementary to the SIM, in the sense of cryptographic functionality.

It is already possible to combine the WIM and SIM functionality on one (multi-application) smartcard; this provides a smooth transition from GSM to 3rd generation mobile communication. Such smartcards may also be interesting candidates for the SHAMAN security module. In the following paragraph, it will be investigated how much of the SM reference model can be covered by them.

5.2 Coverage of SM reference model

A comparison with section 4.2.1 shows that the basic functions for the intermediate security level are fulfilled by a WIM-standard compliant SM.

Conflicts could arise, for example, if the SHAMAN-SM demands for the implementation of a different hash algorithm than SHA-1, but this would probably be of minor importance.

A WIM could also meet some of the requirements for the "high" level of section 4.1.1: The storage of short-term keys is, of course, no additional difficulty if it is already possible to store long-term keys (only a matter of storage space). The on-card key-pair generation is provided by some WIM implementations (for example, the StarWIM™ [7]). Diffie-Hellman computations also belong to the abilities of a WIM.

This means that a WIM already possesses some of the required abilities; however, some things are still missing as described in the following paragraph.

5.3 What is missing?

Although the basic functions are covered, even for the intermediate-level security some additional functionality to support the SHAMAN protocols - as described in 4.2.1 - would probably have to be implemented. Preliminary ideas about the realisation have already been given in section 4.3.

The personal-CA-level functionality, which mainly deals with certificate handling, is not commonly realized in today's tamper-resistant devices, due to their limited capacity. However, dedicated personal-CA smartcards could be feasible today.

5.4 References

[1] Subscriber Identity Modules (SIM), Functional Characteristics, ETS 300 509 (GSM 02.17)

[2] SIMAlliance, see www.simalliance.org/simalliance/default.asp

[3] WAP Identity Module Specification, WAP-198-WIM, see www.wapforum.org

[4] USIM and IC card requirements, 3GPP specification 21.111, see www.3gpp.org/specs/titles-numbers.htm

[5] Wireless Transport Layer Security Specification, WAP-199-WTLS, see www.wapforum.org

[6] PKCS #15 v1.0: Cryptographic Token Information Format Standard, RSA Laboratories, see www.rsasecurity.com/rsalabs/pkcs/index.html

[7] STARWIM - Introducing PKI to the mobile world, product data sheet No. 2882328, Giesecke&Devrient GmbH, München, 2002

6 Conclusion and open issues

In section 3.1 we considered how the data and functions required to support network access security should be distributed between the terminal equipment and the security module. A functionality split was studied for one secret key authentication protocol and one public key protocol. In addition, the involvement of the security module in security mechanisms for protecting the confidentiality and integrity of the access link were studied. Based on this work some general principles were proposed on how to determine the security functionality split between the security module and the terminal equipment.

The work on public key protocols focused on the IKE protocol. In future work it is proposed to investigate the functionality split for other public key protocols that are being investigated by SHAMAN WP1. In particular, the protocols that are likely to be implemented in the demonstrator as part of SHAMAN WP5 will be studied with the highest priority to ensure that the most appropriate functionality split scenarios are implemented within the project. In particular, son-of-IKE candidate protocols studied within SHAMAN WP1 such as JFK will be considered. Also with regard to public key protocols, the role of the security module in revocation checking of certificates that may be used by the mobile node side will be considered in future work.

In the work so far we have assumed that the security module is not directly involved in the protection of the access link communications and that instead the session keys are generated as part of authentication on the security module and then passed to the terminal where they are used. In this scenario the terminal is trusted to perform certain security operations. Further work will involve studying threats associated with un-trusted terminals to determine whether additional mechanisms and techniques involving the security module are required. The work will also consider the role of the security module in security association establishment.

In section 3.2 we considered secure PAN internal communication and secure download. It was noticed that every pair of first party components must have the ability to make authenticated key exchange without any user interaction. This is possible because these components are imprinted. It was studied how the SM is involved in the imprinting process. The ,minimum requirement for the SM is that the shared initial secret must be stored on the SM and that no information that may reveal this secret should be given out of the SM.

After the components are imprinted they can establish a connection, which is authenticated and integrity protected. It was identified that the authentication process should be done in the SM, and therefore the SM needs an authentication algorithm. Confidentiality and integrity protection on the SM is probably not practical; therefore the SM only provides key to the terminal that are derived from the initial secret by a one-way function. It was identified that the SM can support identification and location privacy by computing responses for random challenges.

A list of required functionality for the SM was presented. It should be part of future work to identify what algorithms the SM must support.

In section 3.2.3 it was described that the SM may contain public keys of trusted root CAs, because origin of downloaded content must authenticated. The SM may also verify the certificate, which is provided along with the downloaded content. It is proposed as future work to study whether SM may support certificate revocation and checking policies.

In section 3.4 it was studied how the workload of security functions on a SM can be distributed. The first approach is to distribute the SM into a group of SMs and the second approach is that the security functionality is distributed between the SM and an untrusted device.

The distribution of a single private key operation between the SM and an untrusted device was described. The chosen public key algorithm is RSA, because it is the most used public key algorithm and all the biggest CAs support it. A rough estimation is that the workload for the SM in the distributed case is 7% compared to a standard RSA computation. It is proposed as a further study item to give an estimate for full performance of a distributed RSA computation, which contains message transfers between SM and terminal. It is also proposed as a further study item to consider what is the difference between workloads, when the SM has a co-processor for RSA.

From the viewpoint of the SM-implementation on a smartcard, in addition to conventional features some new features have been introduced, especially the protocol support for e.g. IKE and AAA for IPv6. Similar functionality will possibly become more common in the near future. Once enough experience has been collected, it is possible that the work done in SHAMAN could influence corresponding smartcard standards.

Distributing the workload between the SM and the terminal enables security features today which would otherwise not be possible. Further research on this will be useful, although with increasing resources in the smartcard more of the workload can be shifted to the smartcard itself.

Existing smartcards, like implementations of the WIM standard, already offer much of the functionality which is required for a SHAMAN security module. The protocol support must be added to be fully compliant. At the present time, it is useful to choose subsets out of the full list of SM requirements which can be implemented on today’s devices. Therefore, several security levels have been defined.

The final deliverable in WP4 will give more details about hardware specification and implementation guidelines.

7 Appendix

7.1 Annex A: AAA for IPv6

Key to symbols used in the message flows:

SM	Security module (e.g. removable smart card)
TE	Terminal equipment
MN	Mobile Node (=SM+TE)
AN	Access network
HN	Home network
ID	User identity
Ka	Long term secret authentication key shared between HN and USIM
Ks	Session key agreed between AN and MN as a result of the protocol
N_1	Authentication challenge generated by AN
N_2	Authentication challenge generated by USIM
N_3	Authentication challenge generated by HN
N_4	Authentication challenge generated by HN during the current protocol run
N_4’	Authentication challenge generated by HN during the previous protocol run
(X)AUTN1	(Expected) authentication response for authenticating USIM
(X)AUTN2	(Expected) authentication response for authenticating HN
F1_Ka	Authentication algorithm for USIM authentication using Ka as the key
F2_Ka	Authentication algorithm for HN authentication using Ka as the key
F3_Ka	Session key agreement algorithm using Ka as the key

Basic protocol

SM		TE		AN		HN
				Generate random N_1		
			<-	**N_1**		
	<-	**N_1**				
Generate random N_2 Read ID AUTN1=F1_Ka(N_1, N_2, ID)						
N_2, ID, AUTN1	->					
		N_1, N_2, ID, AUTN1	->			
				N_1, N_2, ID, AUTN1	->	
						XAUTN1=F1_Ka(N_1, N_2, ID) If AUTN1=XAUTN1: AUTN2=F2_Ka(N_2, N_1, ID) Generate random N_3 Ks=F2_Ka(N_3, AUTN2)
					<-	**N_3, AUTN2, Ks**
				Store Ks		
			<-	**N_3, AUTN2**		
	<-	**N_3, AUTN2**				
XAUTN2=F2_Ka(N_2, N_1, ID) If XAUTN2=AUTN2: Ks=F3_Ka(N_3, AUTN2)						
Ks	->					
		Store Ks				

Basic protocol with an additional HN challenge

SM		TE		AN		HN
				Generate random N_1		
			<-	**N_1**		
	<-	**N_1**				
Generate random N_2 Read ID and N_4’ AUTN1=F1_Ka(ID, N_1, N_2, N_4’)						
N_2, N_4’, ID, AUTN1	->					
		N_1, N_2, N_4’, ID, AUTN1	->			
				N_1, N_2, N_4’ ID, AUTN1	->	
						XAUTN1=F1_Ka(N_1, N_2, N_4’, ID) If AUTN1=XAUTN1: AUTN2=F2_Ka(N_2, N_1, ID) Generate random N_3 Generate random N_4 Ks=F2_Ka(N_3, AUTN2)
					<-	**N_3, N_4, AUTN2, Ks**
				Store Ks		
			<-	**N_3, N_4, AUTN2**		
	<-	**N_3, N_4, AUTN2**				
XAUTN2=F2_Ka(N_2, N_1, ID) If XAUTN2=AUTN2: Ks=F3_Ka(N_3, AUTN2) Set N_4’= N_4						
Ks	->					
		Store Ks				

7.2 Annex B: Distributed IKE

7.2.1 Abbreviations

CKY-x	If x is I it is initiators cookie and if x is R it is responder's cookie
HDR	ISAKMP header, when written HDR* it indicates payload encryption
IKE	Internet Key Exchange
KE	Key exchange payload
ME	Mobile equipment
MS	Mobile Station
<P>_b	Body of payload <P>
PFS	Perfect Forward Secrecy
SAi_b	Entire body of the SA payload (minus the ISAKMP generic header)
SKEYID	String derived from secret material
SKEYID_e	Keying material used to protect confidentiality of ISAKMP messages
SKEYID_a	Keying material used to authenticate ISAKMP messages
SKEYID_d	Keying material used to derive keys for non-ISAKMP security associations
SN	Serving Network
TRD	Tamper Resistant Device

7.2.2 Overview of IKE

The Internet Key Exchange (IKE) protocol that is described in RFC2409 is hybrid protocol. Negotiation of IPSec SA with IKE is done in two phases. In the first phase parties create a bi-directional IKE (ISAKMP) SA. By using this IKE SA the parties will have a secure and authenticated channel and they will also have keying material for an IPSec SA, i.e. the string SKEYID_d. In phase 1 the parties first negotiate what algorithms to use, make a Diffie-Hellman key exchange, and finally perform mutual authentication. Phase 1 can be done in two modes namely main mode or aggressive mode. Main mode MUST be implemented and aggressive mode SHOULD be implemented. After phase 1 comes phase 2, where parties can negotiate one or more new IPSec SAs by using quick mode. According to the RFC, quick mode MUST be implemented. In quick mode parties will negotiate the algorithms that are used and if PFS (perfect forward secrecy) is required then there is the possibility to make an additional Diffie-Hellman key exchange. Authentication is done by sending hashes that are derived from SKEYID_a, and all messages that are sent are encrypted by a key based on SKEYID_e. After phase 1 it is also possible to negotiate new group parameters for subsequent Diffie-Hellman key exchanges. New group mode SHOULD be implemented. In addition after phase 1 informational exchanges can occur, for example to notify that an error has occurred. All messages in IKE are ISAKMP messages.

This section describes the requirements for distributing IKE between tamper resistant device (TRD), e.g. USIM or WIM and ME. The most important requirement is that IPSec SAs cannot be created without a TRD. The IKE protocol will run on an ME and some parts of the calculation will be done on a TRD. These calculations depend on what authentication methods are used. If ME gets g^xy, then ME or some attacker that has access to the ME can derive SKEYID and all authenticated keying material. In the following it is listed how secret strings are derived:

For signatures:	SKEYID=prf(Ni_b|Nr_b,g^xy)
For public key encryption:	SKEYID=prf(hash(Ni_b|Nr_b),CKY-I|CKY-R)

For pre-shared keys: SKEYID=prf(pre-shared-key,Ni_b|Nr_b)

SKEYID is really a secret string also in case of public key encryption because public key encryption has been applied on nonces Ni_b and Nr_b. So in phase 1 active parties share the secret SKEYID and they mutually authenticate. This authentication uses the following hashes:

HASH_I=prf(SKEYID,g^xi|g^xr|CKY-I|CKY-R|SAi_b|IDii_b)

HASH_R=prf(SKEYID,g^xr|g^xi|CKY-R|CKY-I|SAi_b|IDir_b)

The result of phase 1 is the following authenticated keying material:

SKEYID_d=prf(SKEYID,g^xy|CKY-I|CKY-R,|0)

SKEYID_a=prf(SKEYID,SKEYID_d|g^xy|CKY-I|CKY-R,|1)

SKEYID_e=prf(SKEYID,SKEYID_a|g^xy|CKY-I|CKY-R,|2)

So if SKEYID or g^xy is revealed outside the TRD, then it is possible that an ME can run phase 2 without a TRD and therefore create new IPSec SAs itself. In this case it is still possible that the IKE SA has been created in a way that is authorised by the TRD. This means that phase 1 authentication must be done on the TRD and this must be the minimum requirement for the TRD. In addition it should be possible for the TRD to check the validity of SN authentication. In next section we look closer at these scenarios, because they depend on the authentication method. Every phase 1 authentication method is divided in two scenarios namely simple and complicated. In the simple scenario TRD is required in mutual authentication so after phase 1 the ME can create IPSec SAs without the TRD. In the complicated scenario SKEYID is not revealed to the ME and so IPSec SAs cannot be created without the TRD.

7.2.3 Different authentication methods of IKE

We look first at the requirements on the TRD in phase 1.

7.2.3.1 Authentication with signatures

<u>7.2.3.1.1 Simpler Scenario</u>

Let's assume first that initiator is the MS and the responder is the SN. Now the proposal of the initiator must contain only those signature algorithms and encryption algorithm for IDENT_I, that are supported by the TRD. First the simpler scenario is presented where SKEYID is given to the ME. Main mode is described as follows:

```
TRD                 initiator                  Responder


                    HDR, SA         -->
                                    <--        HDR,SA
                    HDR, KE, Ni     -->
                                    <--        HDR, KE, Nr
            <--     HASH_I
```

```
Increase COUNTS
By one
If COUNTS<BOUNDS
SIG_I [,IDENT_I]-->
                      HDR*, IDii, [ CERT, ] SIG_I -->
                                                <--  HDR*, IDir, [ CERT, ] SIG_R
                 <--  SIG_R, HASH_R
If SIG_R is
valid then set
COUNTS=0
Else
terminate
```

Our idea is to put a counter COUNTS, which counts the number of signatures that are generated by TRD. Then there are upper bound BOUNDS that COUNTS cannot exceed. The reason for COUNTS is that otherwise the terminal can ask TRD to sign a large number of signatures and get some extra information about the signing key in the TRD. This also gives protection against DPA (differential power analysis) attacks.

This document proposes that the TRD should give Identification Data on the Identification Payload IDii, it is denoted by IDENT_I. To guarantee that the identity is from the TRD, it should be encrypted by the responder's public key. This IDENT_I can be for example the IMSI and in Identification Payload the ID Type is ID_KEY_ID (see RFC 2407).

In this scenario the TRD must contain algorithms for signature, DSS signatures, RSA signatures or both of them. So the ability to calculate modular powers of big integers and hashes and some encoding functions are needed in the TRD. Signing verification requires two PK operations on the TRD and if encryption of IDENT_I is required the total number of PK operations is three. Later we present some approximations on the sizes of required c-code for these algorithms.

Aggressive mode is described as follows:

```
TRD                   Initiator                        Responder


                <--   [Request for IDENT_I]
 [IDENT_I]      -->
                      HDR, SA, KE, Ni, IDii      -->
                                                 <--   HDR, SA, KE, Nr, IDir,
                                                       [ CERT, ] SIG_R
              <--     HASH_I, HASH_R,  SIG_R
Verify SIG_R
SIG_I         -->
                      HDR, [CERT, ] SIG_I         -->
```

Note that BOUNDS is not needed because the TRD first verifues the SIG_R before it reveals the SIG_I. We propose that IDENT_I should be encrypted by the responder's public key. In this case three PK operations are needed. The required sizes of the algorithms in the TRD are approximately the same as in main modealthough the number of ISAKMP messages between initiator and responder has significantly reduced. Unlike main mode no ISAKMP messages are encrypted so aggressive mode doesn't secure the identities from outsiders, but if IDENT_I is provided by the TRD and is encrypted then the identity of the initiator is protected.

Next we look at the situation where the initiator is the SN and the responder is the MS. In this case the proposal of the initiator must contain only those signature algorithms and encryption algorithms for IDENT_I, that are supported in the TRD. Main mode is described as follows:

```
Initiator                    Responder                 TRD

HDR, SA               -->
                     <--     HDR, SA
HDR, KE, Ni           -->
                      <--    HDR, KE, NR
HDR*, IDii, [CERT, ]
SIG_I                 -->
                             HASH_I, HASH_R,
                             SIG_I              -->
                                                       Verifies SIG_I,
                                                       If valid
                                                <--    SIG_R [, IDENT_I]
                             HDR*, IDir, [CERT, ]
                      <--    SIG_R
```

As seen above, this case doesn't put any extra requirements on the TRD compared to the previous case. Note that COUNTS is not needed because the TRD first checks the validity of the other party's signature before it reveals SIG_R.

Aggressive mode is described as follows:

```
Initiator                          Responder                TRD

HDR, SA, KE, NI, IDii       -->
                                   HASH_R            -->
                                                            Increase COUNTS
                                                            By one
                                                            If COUNTS<BOUNDS
                                                     <--   SIG_R [,IDENT_I
                   <--      HDR, SA, KE, Nr,
                             IDir, {CERT, ] SIG_R
HDR, [CERT, ] SIG_I       -->
                                   HASH_I, SIG_I              -->
                                                            If SIG_R is
                                                            valid then set
                                                            COUNTS=0
                                                            Else terminate
```

Note that COUNTS is needed because otherwise the ME could send a lot of signing requests to the TRD.

7.2.3.1.2 Complicated scenario

In this scenario, quick mode can be run only by using the TRD. First we assume that the initiator is the MS. Main mode is described as follows:

```
TRD                      initiator                      Responder

                         HDR, SA              -->
                                              <--       HDR, SA
                  <--    Request for g^x
Increase COUNTR
by one. If
COUNTR<BOUNDR
Generate pseudo
random x and Ni
g^x,Ni.           -->
                         HDR, KE, Ni          -->
                                              <--       HDR,  KE, Nr
                  <--    g^y, Nr, CKY-I, CKY-R,
                         SAi_b
If COUNTS<BOUNDS
then
Calculate (g^y)^x,
SKEYID, SKEYID_d,
SKEYID_a, SKEYID_e,
HASH_I and HASH_R.
```

Derive symmetric key
K from SKEYID_e.
Encrypt IDii, [CERT,]
SIG_I without header,
By using K. Let MES_I
Be encrypted message

```
MES_I          -->
                     HDR, MES_I          -->   Lets denote MES_R
                                               Datagram IDir, [CERT, ]
                                               SIG_R, encrypted by K
                                         <--   HDR, MES_R
             <--      MES_R
Decrypt MES_R,
Verify SIG_R,
if SIG_R is valid set
COUNTR=0, COUNTS=0.
IDir           -->
                     Verify IDir
```

Note that the ME cannot perform a man-in-the-middle attack, because if it sends some g^z to the TRD, then the TRD gives out a false SIG_I which is based on g^x and g^z. If this happens the responder would then try to verify this false SIG_I and notice that this signature is not based on g^y and is therefore not accepted. As the ME could get some statistical data if it can send lots of false g^y's, COUNTS is needed. Because MES_I is encrypted, there is no need for encrypting IDENT_I. In this scenario there is also a counter COUNTR that counts generated pseudorandom numbers. That is done to avoid an attacker getting too much information about the seed of the pseudorandom function (prf). The TRD must be capable of calculating modular powers of big integers, hashes and proper encoding methods, like in the simpler scenario. For this scenario a prf and a symmetric key cipher is needed. The TRD should also store the seed of the prf, g^y, SKEYID, SKEYID_d, SKEYID_a, SKEYID_e[, CERT] and K. Now four calculations of modular powers with big integers are required in the TRD. These are calculating g^x, (g^y)^x, SIG_I and verifying SIG_R.

Aggressive mode is describe as follows:

```
TRD                      initiator                          Responder


                     <--  Request for phase 1
Increase COUNTR
By one, calculate
Pseudorandom x and
Ni, g^x.
```

g^x, Ni[, IDENT_I]	-->			
		HDR, SA, KE, Ni, IDii	-->	
			<--	HDR, SA, KE, Nr, IDir, [CERT,] SIG_R
	<--	g^y, Nr, CKY-I, CKY-R, [IDii,] IDir, SAi_b [, CERT], SIG_R		
Calculate (g^x)^y, HASH_I, verify SIG_R. If SIG_R is valid set COUNTR to zero calculate HASH_R, SIG_R, SKEYID, SKEYID_d, SKEYID_a, SKEYID_e.				
[CERT,] SIG_I	-->			
		HDR [, CERT] , SIG_I	-->	

The requirements in this aggressive mode are similar to main mode, except that a symmetric cipher is not needed. Note that the ME get IDir directly, because it is not encrypted.

Next we look at the case where MS is the responder. Main mode is described as follows:

Initiator		Responder		TRD
HDR, SA	-->			
	<--	HDR, SA		
HDR, KE, Ni	-->			
		g^x, Ni	-->	
				Increase COUNTR by one, if COUNTR<BOUNDR then generate pseudorandom y and Nr. Calculate g^y.
			<--	g^y, Nr
	<--	HDR, KE, Nr		
HDR, MES_I	-->			
		MES_I, CKY-I, CKY-R, Sai_b	-->	
				Calculate g^xy,

				SKEIYID, SKEYID_d, SKEYID_a, SKEYID_e. Derive symmetric key K from SKEYID_d. Decrypt MES_I with K, calculate HASH_I and verify SIG_I if it is valid Calculate HASH_R and SIG_R. Encrypt datagram IDir, [CERT,] SIG_R, with K. Set COUNTR=0.
			<--	MES_R
	<--	HDR, MES_R		

The requirements for the TRD are the same as when MS is the initiator.

Aggressive mode is described as follows:

Initiator		Responder		TRD
HDR, SA, KE, Ni, IDii	-->			
		g^x, Ni, IDii, SAib, CKY-I, CKY-R [,IDir]	-->	
				Increase COUNTS by one If COUNTS<BOUNS, then Generate pseudo random y and Nr. Calculate g^y, g^xy, calculate SKEYID, SKEYID_d, SKEYID_a, SKEYID_e, HASH_R and SIG_R.
			<--	SIG_R [,IDENT_R]
	<--	HDR, SA, KE, Nr, IDir, [CERT,] SIG_R		
HDR, [CERT,] SIG_I	-->			
		SIG_I	-->	

				Calculate HASH_I and verify SIG_I. If SIG_I is valid set
				COUNTS=0.

7.2.3.2 Authentication with Public Key Encryption

In this case authentication is done using public key encryption of nonces. This method has also a modified version, named Authentication with Revised Mode of Public Key Encryption. We first look at simpler scenarios as in section 2.1. Then we look at what the requirements are in the complicated scenario.

7.2.3.2.1 Simpler scenario

In this scenario the private key is stored on the TRD and therefore without it the ME cannot create an IKE SA. Let's first assume that the initiator is the MS. Main mode is described as follows:

TRD		initiator		Responder
		HDR, SA	-->	
			<--	HDR, SA
	<--	[Identity request, for responders public key]		
[<IDii_b>Pubkey_r]	-->			
		HDR, KE, [HASH(1),] <IDii_b>Pubkey_r, <Ni_b>Pubkey_r	-->	
			<--	HDR, KE, <IDir_b>Pubkey_i, <Nr_b>PubKey_i
	<--	<IIDir_b>PubKey_i, <Nr_b>PubKey_i		
Increase COUNTD by one And if COUNTD<BOUNDD Decrypt <IDir_b>Pubkey_i and <Nr_b>PubKey_i				
Nr_b, IDii_b	-->			
		HDR*, HASH_I	-->	
			<--	HDR*, HASH_R
	<--	HASH_R, SKEYID, g^x, g^y,		

		CKY-R, CKY-I, SAi_b
Calculate		
HASH_R and		
compare it to		
Given HASH_R.		
If it is valid set		
COUNTD=0		
else terminate.		

The counter COUNTD has a similar meaning to the COUNTS in signature-based authentication. In this case where Nr_b is revealed to the ME, it can calculate HASH_I, but the ME cannot calculate HASH_R because only the TRD knows IDir_b. Here two Public Key operations for TRD are needed. If the identity of the TRD should be protected, then three PK operations are needed, one extra for <IDii>PubKey_r. So TRD must be able to calculate modular powers of big integers, for the decryption of <Nr_b>PubKey_I. The TRD must also be capable of calculating hashes. If a whole RSA encryption in PKCS #1 format is required, then the TRD must have the ability to calculate some encoding methods.

Aggressive mode is described as follows:

TRD		initiator		Responder
	<--	[Identity request,		
		for responders		
		public key]		
[<IDii_b>Pubkey_r]	-->			
		HDR, SA, [HASH(1),] KE,		
		<IDii_b>PubKey_r,		
		<Ni_b>PubKey_r	-->	
				HDR, KE, <IDir_b>Pubkey_i,
			<--	<Nr_b>PubKey_I, HASH_R
	<--	<Nr_b>PubKey_I		
Increase				
COUNTD by one				
And if COUNTD<BOUNDD				
Decrypt				
<IDii_b>Pubkey_r				
and				
<Nr_b>PubKey_I				

Nr_b, IDii_b	-->			
	<--	SKEYID, g^x, g^y, CKY-I,		
		CKY-R, SAi_b, HASH_R		
Calculate				
HASH_R and				
compare it to				
Given HASH_R.				
If it is valid set				
COUNTD=0				
else terminate.				
Send OK respond	-->			
		HDR*, HASH_I	-->	

Here requirements are the same as in main mode. One should note that although HASH_I and HASH_R are sent as plain text, identities ID_ix are encrypted.

Next we look at the situation where the responder is the MS. Main mode is described as follows:

Initiator		Responder		TRD
HDR, SA	-->			
	<--	HDR, SA		
HDR, SA, [HASH(1),] KE,				
<IDii_b>PubKey_r,				
<NI_b>PubKey_r	-->			
		<Ni_b>PubKey_r	-->	
				Increase
				COUNTD by one
				And if COUNTD<BOUNDD
				Decrypt
				<IDii_b>PubKey_I
				and
				<Ni_b>PubKey_r
			<--	Ni_b [,<IDir>PubKey_I]
		HDR, KE, <IDir>PubKey_I,		
	<--	<Nr_b>PubKey_I		
HDR*, HASH_I	-->			
		SKEYID, g^x, g^y, CKY-I,		

		CKY-R, SAi_b, HASH_R	-->	
				Calculate
				HASH_R and
				compare it to
				Given HASH_R.
				If it is valid set
				COUNTD=0
				else terminate.
			<--	Send OK respond
	<--	HDR*, HASH_R		

Again requirements for the TRD are the same as in situation where MS is initiator. Aggressive mode is described as follows:

Initiator		Responder		TRD
HDR, SA, [HASH(1),] KE,				
<IDii_b>PubKey_r,				
<NI_b>PubKey_r	-->			
		<Ni_b>PubKey_r	-->	
				Increase
				COUNTD by one
				And if COUNTD<BOUNDD
				Decrypt
				<IDii_b>PubKey_I
				and
				<Ni_b>PubKey_r
			<--	Ni_b[,<IDir>PubKey_I]
		HDR, KE, <IDir>PubKey_I,		
	<--	<Nr_b>PubKey_I		
HDR, HASH_I	-->			
		SKEYID, g^x, g^y, CKY-I,		
		CKY-R, SAi_b, HASH_R	-->	
				Calculate
				HASH_R and
				compare it to

				Given HASH_R.
				If it is valid set
				COUNTD=0
				else terminate.
			<--	Send OK respond
	<--	HDR, HASH_R		

Requirements for the TRD are the same as in main mode.

7.2.3.2.2 Complicated scenario

In this scenario, an IKE SA is created but not revealed to the ME. This scenario has the advantage that the ME cannot run quick mode without the TRD.

Main mode is described as follows:

TRD		initiator		Responder
		HDR, SA	-->	
			<--	HDR, SA
	<--	Request for nonce		
<Ni_b>Pubkey_r [,<IDii_b>Pubkey_r]	-->			
		HDR, KE, [HASH(1),] <IDii_b>Pubkey_r, <Ni_b>Pubkey_r	-->	
			<--	HDR, KE, <IDir_b>Pubkey_i, <Nr_b>PubKey_i
	<--	g^x, g^y, CKY-I, CKY-R, <Nr_b>PubKey_I		
Increase COUNTD by one And if COUNTD<BOUNDD Decrypt <Nr_b>PubKey_I and calculate				

SKEYID, SKEYID_d, SKEYID_a, SKEYID_e HASH_I. Derive key K from SKEYID_d				
<HASH_I>K	-->			
		HDR*, HASH_I	-->	
			<--	HDR*, HASH_R
	<--	<HASH_R>K, g^y		
Calculate HASH_R and decrypt <HASH_R>K, if these are same set COUNTD=0, else terminate.				

Requirements for the TRD are the same as in simpler scenario plus symmetric key cipher must be in TRD and one extra PK operation in the TRD for Nr_b is required. Also the TRD must store SKEYID, SKEYID_d, SKEYID_a, SKEYID_e.

Aggressive mode is described as follows:

TRD		initiator		Responder
	<--	Request for nonce		
<Ni_b>PubKey_r, [<IDii_b>Pubkey_r]	-->			
		HDR, SA, [HASH(1),] KE, <IDii_b>PubKey_r, <Ni_b>PubKey_r	-->	
			<--	HDR, KE, <IDir_b>Pubkey_i, <Nr_b>PubKey_I, HASH_R
	<--	<Nr_b>PubKey_I, g^x,g^y, CKY-R, SAi_b, HASH_R		
Increase COUNTD by one And if COUNTD<BOUNDD Decrypt <IDii_b>Pubkey_r and				

<Nr_b>PubKey_I			
Calculate			
SKEYID, SKEYID_d,			
SKEYID_e, SKEYID_a,			
HASH_R and			
compare it to			
given HASH_R.			
If it is valid set			
COUNTD=0			
else terminate.			
Send OK respond	-->		
		HDR*, HASH_I	-->

Requirements are the same as in main mode except that a symmetric key cipher is not needed. Next we look at the case where the MS is the responder. Main mode is described as follows:

Initiator		Responder		TRD
HDR, SA	-->			
	<--	HDR, SA		
HDR, SA, [HASH(1),] KE,				
<IDii_b>PubKey_r,				
<Ni_b>PubKey_r	-->			
		Request for nonce	-->	
				[<IDir>PubKey_I,]
			<--	<Nr_b>PubKey_I
		HDR, KE, <IDir>PubKey_i,		
	<--	<Nr_b>PubKey_i		
HDR, HASH_I	-->			
		<IDii_b>PubKey_r,		
		<Ni_b>PubKey_r ,		
		g^x, g^y, CKY-I,		
		CKY-R, SAi_b, HASH_I -->		
				Increase
				COUNTD by one
				And if COUNTD<BOUNDD
				Decrypt

```
                                                        <IDii_b>PubKey_i
                                                        and
                                                        <Ni_b>PubKey_r
                                                        Calculate
                                                        HASH_I and
                                                        compare it to
                                                        Given HASH_I.
                                                        If it is valid set
                                                        COUNTD=0
                                                        else terminate.
                                                    <-- Send OK responds
                              <--   HDR, HASH_R
```

Requirements for the TRD are the same as in the situation where the MS was the initiator.

Aggressive mode is described as follows:

```
Initiator                           Responder                   TRD


HDR, SA, [HASH(1), ] KE,
<IDii_b>PubKey_r,
<NI_b>PubKey_r                -->
                                    <Ni_b>PubKey_r     -->
                                                        Increase
                                                        COUNTD by one
                                                        And                 if
                                                        COUNTD<BOUNDD
                                                        Decrypt
                                                        <IDii_b>PubKey_I
                                                        and
                                                        <Ni_b>PubKey_r
                                                    <--  Ni_b[,<IDir>PubKey_I]
                                    [Identity request,
                                    for responders
                                    public key]             -->
                                                    <--  [<IDir>PubKey_I]

                                    HDR, KE, <IDir>PubKey_i,
                              <--   <Nr_b>PubKey_i
```

```
HDR, HASH_I                -->
                               SKEYID, g^x, g^y, CKY-I,
                               CKY-R, SAi_b, HASH_R   -->
                                                         Calculate
                                                         HASH_R and
                                                         compare it to
                                                         Given HASH_R.
                                                         If it is valid set
                                                         COUNTD=0
                                                         else terminate.
                                                     <-- Send OK responds


                          <--  HDR, HASH_R
```

Requirements for the TRD are the same as main mode.

7.2.3.3 Authentication with Revised Mode of Public Key Encryption

Authentication with Public Key has that drawback that it needs four PK operations. So especially in the complicated scenario this can be a problem, because the TRD don't have great computational capacity. The idea of revised mode is simply that PK encryption of IDix is replaced by symmetric key encryption of IDix. So three PK operations would be adequate. One should notice that this would help TRD only when IDix is protected by TRD.

7.2.3.4 Authentication with Pre-Shared Key

The idea in this mode is that the TRD contains a pre-shared key that is not exposed to anybody. So without the TRD the ME can't authenticate. As in previous cases we first look at the simple scenario.

7.2.3.4.1 Simpler scenario

We first assume that the ME is the initiator. Main mode is described as follows:

```
TRD                     initiator                    Responder


                        HDR, SA                -->
                                               <--   HDR, SA
                        HDR, KE, Ni            -->
                                               <--   HDR, KE, Nr
                   <--  Ni, Nr,  g^x, g^y, CKY-I,
                        CKY-R, SAi_b
Increase COUNTP
by one, if
COUNTP<BOUNDP
```

calculate SKEYID,				
SKEYID_d, SKEYID_a,				
SKEYID_e.				
SKEYID_d [,IDENT_I]	-->			
		HDR*, IDii, HASH_I	-->	
			<--	HDR*, IDir, HASH_R
	<--	HASH_R		
Calculate HASH_R				
and compare it to				
given HASH_R, if				
it is valid then set				
COUNTP=0				
SKEYID	-->			

This method has very light requirements especially for the TRD. Both the TRD and the ME don't need any PK operations. One PK operation for TRD is needed, if IDENT_I is sent in encrypted form. Here the TRD must calculate SKEYID and HASH_R; these are simply calculating a prf, i.e. HMAC. SKEYID_d can be given to the ME, because the ME can't derive SKEYID from SKEYID_d.

Aggressive mode is described as follows:

TRD		initiator		Responder
		[Identity request,		
		for responders		
	<--	Public key]		
[IDENT_I]	-->			
		HDR, SA, KE, Ni, IDii	-->	
			<--	HDR, SA, KE, Nr, IDir, HASH_R
	<--	Ni, Nr, HASH_R		
Increase COUNTP				
by one, if				
COUNTP<BOUNDP				
calculate SKEYID.				
SKEYID	-->			
		g^x, g^y, CKY-I,		
	<--	CKY-R, SAi_b, IDii_b		
Calculate HASH_R				

and compare it to			
given HASH_R, if			
it is valid then set			
COUNTP=0 else			
Terminate.			
Verification OK	-->		
		HDR, HASH_I	-->

Note that here HASH_R is given to the TRD before SKEYID is revealed to the ME. That is because otherwise the ME could cheat the TRD by changing the origin of the responder to obtain an SKEYID for that responder. This scenario needs fewer numbers of prf operations on the TRD.

Let's look at the situation where the MS is the responder. Main mode is described as follows:

Initiator		Responder		TRD
HDR, SA	-->			
	<--	HDR, SA		
HDR, KE, Ni	-->			
	<--	HDR, KE, Nr		
HDR*, IDii, HASH_I	-->			
		Ni, Nr, g^x, g^y, CKY-I,		
		CKY-R, SAi_b	-->	
				Increase COUNTP
				by one, if
				COUNTP<BOUNDP
				Calculate SKEYID,
				SKEYID_d, SKEYID_a,
				SKEYID_e.
			<--	SKEYID_e
		IDii, HASH_I	-->	
				Calculate HASH_I, if
				It same as given one
				then set
				COUNTP=0
			<--	SKEYID[, IDENT_R]
	<--	HDR*, IDir, HASH_R		

Aggressive mode is described as follows:

Initiator		Responder		TRD
HDR, SA, KE, Ni, IDii	-->			
		Ni, Nr, g^x, g^y, CKY-I,		
		CKY-R, SAi_b, IDii	-->	
				Increase COUNTP
				by one, if
				COUNTP<BOUNDP
				Calculate SKEYID,
				HASH_I, HASH_R
			<--	HASH_R [, IDENT_R]
		HDR, SA, KE, Nr, IDir,		
	<--	HASH_R		
HDR, HASH_I	-->			
		HASH_I	-->	
				Calculate HASH_I, if
				It same as given one
				then set
				COUNTP=0
			<--	SKEYID

7.2.3.4.2 Complicated scenario

Here we present a method for doing phase 1 negotiation in a way that the ME doesn't get the SKEYID. First we present a scenario where the MS is the initiator. Main mode is described as follows:

TRD		initiator		Responder
		HDR, SA	-->	
			<--	HDR, SA
		HDR, KE, Ni	-->	
			<--	HDR, KE, Nr
	<--	Ni, Nr, g^x, g^y, CKY-I,		
		CKY-R, SAi_b		
Increase COUNTP				
by one, if				
COUNTP<BOUNDP				
calculate SKEYID,				
SKEYID_d, SKEYID_a,				

SKEYID_e, HASH_I and MES_I which is datagram IDii, HASH_I encrypted by key K. K is derived from SKEYID_e				
MES_I	-->			
		HDR, MES_I	-->	
			<--	HDR, MES_R
	<--	MES_R		
Decrypt MES_R, calculate HASH_R and compare it to given HASH_R, if it is valid then set COUNTP=0				
SKEYID	-->			

An important feature is that PK encryption of IDENT_I is not needed, because ME can't derive SKEYID and therefore the symmetric key K which is used to encrypt the identity. So the TRD only needs a symmetric cipher, a hash function and some encoding methods.

Aggressive mode is described as follows:

TRD		initiator		Responder
	<--	[Identity request, for responders Public key]		
[IDENT_I]	-->			
		HDR, SA, KE, Ni, IDii	-->	
			<--	HDR, SA, KE, Nr, IDir, HASH_R
	<--	Ni, Nr, HASH_R, IDir, g^x, g^y, CKY-I, CKY-R, SAi_b, IDii_b		
Increase COUNTP by one, if COUNTP<BOUNDP calculate SKEYID.				

Calculate HASH_R and compare it to given HASH_R, if it is valid then set COUNTP=0 else Terminate.			
HASH_I	-->		
		HDR, HASH_I	-->

Next we look at the scenario where the MS is the responder. Main mode is described as follows:

Initiator		Responder		TRD
HDR, SA	-->			
	<--	HDR, SA		
HDR, KE, Ni	-->			
	<--	HDR, KE, Nr		
HDR*, IDii, HASH_I	-->			
		Let MES_I be encrypted Datagram IDii,HASH_I. Ni, Nr, g^x, g^y, CKY-I, CKY-R, SAi_b, MES_I	-->	
				Increase COUNTP by one, if COUNTP<BOUNDP Calculate SKEYID, SKEYID_d, SKEYID_a, SKEYID_e. Decrypt MES_I Calculate HASH_I, if It same as given one then set COUNTP=0 Encrypt datagram IDir, HASH_R
			<--	MES_R
	<--	HDR, MES_R		

Aggressive mode is described as follows:

Initiator		Responder		TRD
HDR, SA, KE, Ni, IDii	-->			
		Ni, Nr, g^x, g^y, CKY-I,		
		CKY-R, SAi_b, IDii	-->	
				Increase COUNTP
				by one, if
				COUNTP<BOUNDP
				Calculate SKEYID,
				HASH_I, HASH_R
			<--	HASH_R [, IDENT_R]
		HDR, SA, KE, Nr, IDir,		
	<--	HASH_R		
HDR, HASH_I	-->			
		HASH_I	-->	
				Calculate HASH_I, if
				It same as given one
				then set
				COUNTP=0
				Else terminate.

7.2.3.5 *Quick mode*

After an IKE SA has been created, parties can create IPSec SAs by using the IKE SA. Here we need SKEYID_e for encryption of ISAKMP messages, SKEYID_a for authenticating parties (mutually) and SKEYID_d for creating keying material for IPSec SA. If in phase 1 the simpler scenario is used, then the ME will have SKEYID_e, SKEYID_a and SKEYID_d, so the TRD is not needed and the whole phase 2 can be run on the ME. The interesting question is how the TRD should be used when the complicated scenario has been used. We first look at the situation where the MS is the initiator.

TRD		initiator	Responder
		M-ID, SA, Ni [,KE]	
	<--	[IDci, IDcr]	
Calculate HASH(1).			
Encrypt received			
Datagram added			
HASH(1) by key			

K derived from				
SKEYID_e.				
MES_I	-->			
		HDR, MES_I	-->	
				HDR*, HASH(2), SA, Nr
			<--	[, KE] [, IDci, IDcr]
	<--	MES_R		
Decrypt datagram				
MES_R. Calculate				
HASH(2) and				
Compare to received				
one. If these are				
same calculate				
HASH(3) and				
KEYMAT. Encrypt				
HASH(3) let encrypted				
message be MES(3)				
MES(3), KEYMAT	-->			
		HDR*, MES(3)	-->	

Note that here

KEYMAT=prf(SKEYID_d, protocol | SPI | Ni_b | Nr_b)

so, KEYMAT can be given to the ME without revealing the secret SKEYID_d. Here the TRD must be capable of calculating hashes and symmetric key ciphering. The situation where the MS is the responder is described as follows:

Initiator		Responder		TRD
HDR*, HASH(1), SA, Ni				
[, KE] [, IDci, IDcr]	-->			
		MES_I, Nr, M-ID		
		[,KE] [, IDci, IDcr][, SA]	-->	
				[Select SA]
				Decrypt datagram
				MES_I. Calculate
				HASH(1) and check

				validity of it. If it is not valid then terminate. Calculate HASH(2) And HASH(3) . Encrypt datagram HASH(2), SA, Nr [, KE] [, IDci, IDcr]
			<--	MES_R
	<--	HDR, MES_R		
HDR,MES(3)	-->			
		MES(3)	-->	
				Decrypt MES(3) and verify HASH(3).

Here the optional security association payload means that the TRD may be allowed to choose the SA.

7.2.4 Platform requirements for TRD

In this section we describe the memory requirements for the TRD. It is assumed that the TRD performs only cryptographic algorithms and the rest of the IKE protocol is run in the terminal. These memory requirements are described by showing what is the size of the needed algorithms on a Pentium 2 platform. Next we look at what algorithms are needed in different scenarios.

Scenario	**algorithms**
Signatures Simple	RSA+ bigintegers+ MD5 + HMAC PKCS #1 encoding
Signatures Complicated	Same as simple+ Des+ Triple Des
Public key encryption Simple	Same as with signatures simple

Public key encryption complicated	Same as with signatures complicated
Revised mode of public key encryption Simple/complicated	Same as with signatures complicated
Pre-Shared key Simple	MD5+ HMAC-MD5
Pre-Shared key complicated	MD5+ HMAC-MD5+ Des+ Triple Des

In the above list only the minimum is required so SHA1 is not required, but if one wants to include SHA1 and HMAC-SHA1 then the sizes of their code must be added to the total size. It should be noted that the TRD needs some other functions too, which are not described in this document.

We assume that IKE without cryptographic primitives can be implemented with a code size of 100k byte. By using this assumption the total requirements for both TRD and ME can be presented roughly as follows:

	TRD	ME
Whole IKE in TRD when only Pre-shared keys are used	100k+15k	-
Whole IKE in TRD and no restrictions for authentication Mechanism.	100k+35k	-
Simple scenario and only Pre-shared keys are used	5k	100k+25k
Simple scenario and no restriction for authentication mechanism.	35k	100k+35k
Complicated scenario and only Pre-shared keys are used	15k	100k+25k
Complicated scenario and no restriction for authentication mechanism.	35k	100k+35k
Whole IKE in ME when only Pre-shared keys are used	-	100k+15k
Whole IKE in ME and no restrictions for authentication mechanism.	-	100k+35k

Memory requirements do not vary a lot between scenarios, but it should be noted that efficiency of the TRD is smaller than that of the ME so the speed comparison may vary a lot.