The Inter-Domain Key Exchange Protocol

A Cryptographic Protocol for Fast, Secure Session-Key Establishment and Re-Authentication of Mobile Nodes after Inter-Domain Handovers

Dissertation

zur Erlangung des Doktorgrades

der Mathematisch - Naturwissenschaftlichen Fakultäten

der Georg-August-Universität zu Göttingen

vorgelegt von

Rene Alexander Soltwisch

aus Lübeck

Göttingen 2006

D7

Referent: Prof. Dr. Dieter Hogrefe

Korreferent: Prof. Dr. Bernhard Neumair

Tag der mündlichen Prüfung: 18. Januar 2006

Abstract

This thesis introduces, specifies and evaluates a novel key establishment mechanism to enable seamless authenticated handovers in IP networks called Inter-Domain Key Exchange Protocol (IDKE). The task of the IDKE protocol is to quickly re-establish trust and a shared session-key between the mobile node and the access network. This is implemented after a mobile node's handover by forwarding the session-key from the previous to the new access network. IDKE's major strength is in providing a secured key forwarding even when the two domains initially do not trust each other. The purpose of the transferred key is to secure the access link, thus providing confidentiality, integrity and access control. Generally such keys are obtained from the mobile node's home network, whereas the IDKE protocol forwards the key locally in between access networks via an exclusively established and secured communication channel. This work specifies security properties for authentication and secrecy and verifies the IDKE protocol by model checking. The protocol is modeled by Communication Sequential Processes (CSP); formal security verification is performed by Failure Divergence Refinement (FDR). Furthermore, the function for handling concurrent protocol runs is added to the IDKE protocol. The extended specification is simulated and verified by utilizing the Specification and Description Language (SDL) in order to analyze the robustness and the scalability of the protocol. Finally, the performance is compared to other approaches such as the Global System for Mobile Communications (GSM) and the Wireless Shared Key Exchange Protocol (W-SKE) using the discrete event simulator OPNET Modeler.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. Dr. Hogrefe for his assistance and the possibility to research under such perfect conditions. The work at the Institute for Informatics at the Georg August University of Göttingen has been a very positive experience. I much appreciate the fact that I was allowed to act independently in a trustful atmosphere.

My thanks go out to my colleagues from the Institute of Informatics at the University of Göttingen who have accompanied me throughout the years. A special thank you to Michael Ebner, Xiaoming Fu and Constantin Werner for the useful discussions on security, mobility and other thesis related topics.

I also wish to thank the students Florian Tegeler, Niklas Steinleitner and Dirk Lessner who wrote Bachelor theses, participated in master courses and who wrote papers with me on subjects that strongly related to this thesis. Furthermore, I am most grateful to all the students who participated in my Network Security seminars and who contributed many useful summaries on a wide range of security related topics.

This thesis would not be in its current form without the numerous proof-readers who have spent their spare time correcting typing errors, dealing with missing indexes, etc. and who have given helpful hints on how to improve my thesis. I thank Michael O'Connell, Edith Werner and Anja Wollrath for their efforts.

I am also greatly indebted to my parents for "introducing" me to this world and who together with my grandparents encouraged me to study computer science, without which this thesis would never have seen the light of day.

Table of Contents

List of Figures

Chapter 1

Introduction

Technical progress in computer engineering and the evolution in computer science have improved the capabilities of electronic devices in computation power and storage. The Internet has also been changed into a publicly accessible network providing a vast variety of services driven by commercial interests. Hence, the consumers' viewpoint has changed to the desire for permanent connectivity, ultra-high bandwidth and various services on demand. Service development will need to be based on the paradigm that every user is potentially mobile. It is necessary for the architecture to be able to carry all types of traffic, some requiring stringent *Quality of Service (QoS)* [APF01], others demading only best effort service. Multimedia and *Voice over IP (VoIP)* [HCW01, KKS01, RS98, RS99] involve special requirements on service-parameters such as delay and delay jitter. Such applications may need to reserve bandwidth and set up necessary service guarantees at bottlenecks.

1.1 Problem Statement

In order to protect their commercial interests, such as charging users, service providers require an *Authentication, Authorization and Accounting (AAA)* [RHK+02] infrastructure. They enforce AAA mechanisms to be processed before guaranteeing services or even providing any network access at all. Thus, specific cryptographic protocols for authentication and session-key establishment are utilized. These protocols indirectly control access and guarantee integrity and confidentiality of data. Whenever a user is not known to the network, this procedure involves further parties acting as guarantors and subsequently requires a significant amount of time. As this procedure is always performed prior to actually releasing the requested resources it becomes problematic when switching networks, because handovers again involve authentication procedures. Due to AAA mechanisms, the delay caused by them creates problems when running real-time applications. Hence, it is difficult to handle the competing desires of mobility and security as a combined task.

1.2 Related Work

The *Internet Engineering Task Force (IETF)* has made considerable efforts in separately facing the issues of security and mobility. Security for IP networks is provided by a security protocol suite referred to as *Internet Protocol Security (IPsec)*. IPsec provides encryption and authentication of data while a sub-protocol called *Internet Key Exchange (IKE)* [PK00] protocol provides session-key establishment. *Mobile IP* [Pre02, JPA04] was also created as an IETF approach to provide IP mobility. Furthermore, Mobile IP was enhanced with the aim of reducing handover latency. The IETF SeaMoby working group for instance, suggested a protocol titled *Context Transfer Protocol (CxTP)* [LNP+05] which forwards parameters between the access networks rather than obtaining them from the home network. However, CxTP assumes the case that a secure channel has already been established. This case is referred to as intra-domain handover, whereas an inter-domain handover describes the case in which both networks do not have any common security relationship. These IETF approaches still lack a solution to the problem of adequately combining performance driven enhancements with security, since they either assume security has already been provided or provide security without considering mobility as a factor. All approaches of the research community such as the context transfer support for IP-based mobility management [Geo04] are aware of the inter-domain handover problematic. However, these approaches do not give any concrete solution as to how to establish the required security association prior to performing CxTP. No formal security evaluation and robustness verification of concurrent protocols runs have been performed for a CxTP-based key forwarding approach.

Other approaches such as the GSM [Hei98] and the W-SKE [SBG+03] follow different principals. Both require the home network to be involved in the session-key establishment procedure. Under certain circumstances these non-localized mechanisms may cause huge delays and thus provide lower performance than localized mechanisms.

1.3 Solution Description

The focus of this thesis is on introducing a novel mechanism especially designed for the problematic case of providing authentication and session-key re-establishment after inter-domain handovers. The main concept is based on building a trust relationship between the involved domains by authenticating both the *Mobile Node (MN)* and the new access network for the previous access network. Consequently, a secure connection is established between the two domains in order to transfer the session-key locally. This novel protocol is called the *Inter-Domain Key Exchange (IDKE)* protocol [SFN+04, STH05]. The IDKE protocol involves three entities: the MN, a previous access router from the domain visited before and a new access router belonging to the current domain. This thesis deals with introducing and formally describing the IDKE protocol, specifying all the involved mechanisms and tasks at each entity, as well as explaining all the exchanged messages in detail. In particular, work has been carried out in the following problem areas:

1. Security verification of the IDKE protocol against its properties. Therefore, this thesis gives a formal definition of the protocol, specifies security goals and analyzes the specification by utilizing formal methods. Security properties such as the secrecy of key and agreements for authentication are verified by model checking based on the *Communicating Sequential Processes (CSP)* algebra [RS01].
2. Robustness tests in order to verify the capability of handling concurrent protocol runs caused by fast moving MNs. The IDKE protocol is specified by utilizing the *Specification and Description Language (SDL)* [ITU92, ITU99]. This specification is then used to simulate specific cases caused by concurrent requests- and cancel-messages.
3. Performance simulation of the IDKE protocol in comparison with other approaches. The *OPNET Modeler* [Opn05], a discrete event simulator, is used to estimate the overall protocol delay of the IDKE protocol, the key-establishment-mechanism of the *Global System for Mobile Communications (GSM)* [Hei98] and the *Wireless Shared Key Exchange protocol (W-SKE)* [SBG+03]. The major task of the performance analysis is to figure out under which conditions the IDKE protocol is faster or slower than the other approaches.

The combination of formal security verification, robustness testing and performance evaluation examines the protocol from three different viewpoints. Therefore, the requirement to provide security for mobile devices in a fast manner is expected to be treated appropriately. Specific behavior involved in this localized solution is examined by robustness evaluations in order to finally obtain a suitable session-key establishment protocol for mobile devices.

1.4 The Structure of the Thesis

This thesis consists of seven chapters as illustrated in Figure 1. The content of all following individual chapters is briefly introduced below:

Chapter 2 introduces the basic concepts of mobility, cryptography and cryptographic protocols. Related IETF approaches such as MobileIP and their performance enhancements in providing seamless connectivity are also given.

Chapter 3 covers the core of the study and thus deals with the IDKE protocol, as well as including the security properties and formally specifying the message flow. The basic concept of the protocol is presented and each message is described in detail. Protocol assumptions and protocol goals are defined as pre- and post-conditions forming the basis, against which the protocol is verified.

Security, robustness and performance evaluations are shown independently of each other as denoted by the three pillars in Figure 1:

Chapter 4, the first pillar deals with security and thus introduces security-analyzing approaches and formal proof methods. The focus here is on model checking provided by CSP [Sch99], FDR [FDR99] and Casper [Low97]. IDKE is formally described in the Casper Notation and the analytical results are also presented here. A CSP specification of the IDKE protocol is given in Appendix C.

Chapter 5, the second pillar, discusses robustness and focuses on concurrent protocol runs and unwanted states of the participating nodes. SDL is introduced as a formal method for validating protocol actions. The IDKE protocol is described as a *Message Sequence Chart (MSC)* [ITU96] and has been verified by simulation; the results of which are also given.

Chapter 6 forms the final pillar and concentrates on protocol performance. As the GSM and the W-SKE protocols are the approaches, to which the performance of the IDKE protocol is compared, they have been included in this chapter. The performance has been evaluated by means of the OPNET Modeler that has been applied as a simulation environment. The simulation results are also presented here.

All three pillars include a section for discussion where the results of security, robustness and performance are examined separately.

Chapter 7 summarizes the entire study; a conclusion is provided as well as giving an outlook on future work.

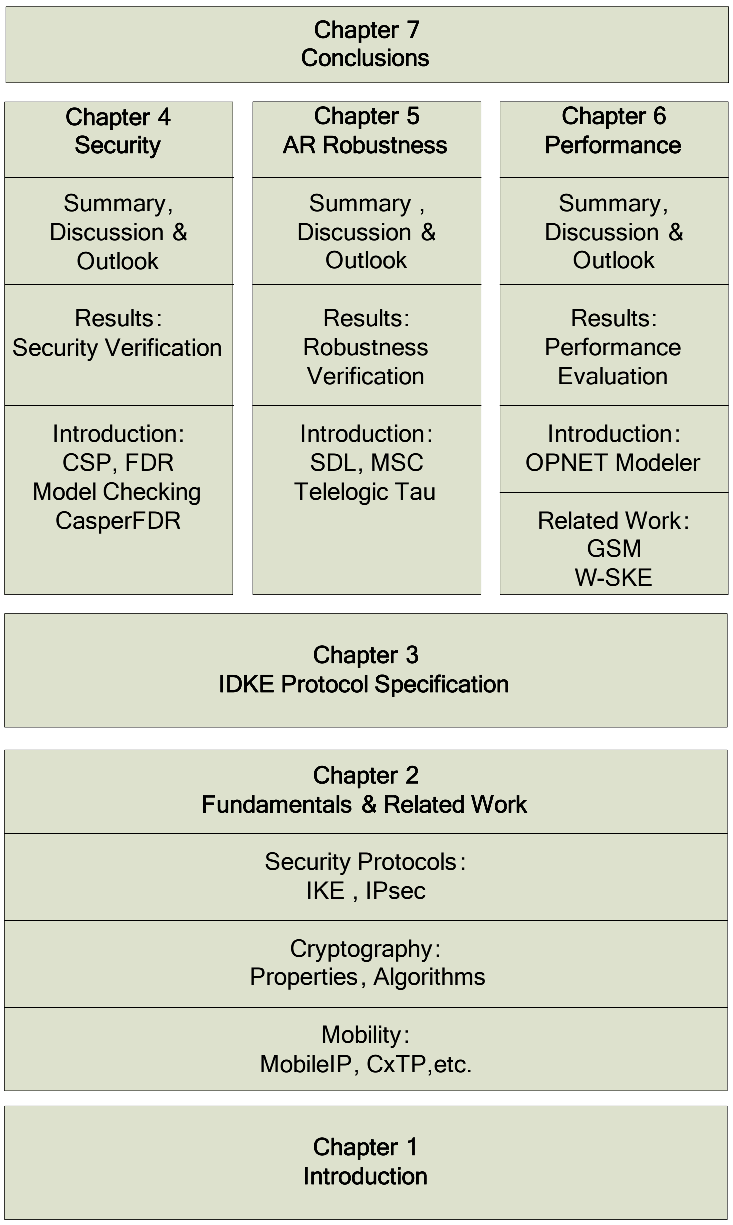

Chapter 7
Conclusions
Chapter 4
Security
Summary, Discussion & Outlook
Results: Security Verification
Introduction: CSP, FDR Model Checking CasperFDR
Chapter 5
AR Robustness
Summary , Discussion & Outlook
Results: Robustness Verification
Introduction: SDL, MSC Telelogic Tau
Chapter 6
Performance
Summary, Discussion & Outlook
Results: Performance Evaluation
Introduction: OPNET Modeler
Related Work: GSM W-SKE
Chapter 3
IDKE Protocol Specification
Chapter 2
Fundamentals & Related Work
Security Protocols: IKE , IPsec
Cryptography: Properties, Algorithms
Mobility: MobileIP, CxTP,etc.
Chapter 1
Introduction

Figure 1: Structure of the Study

Chapter 2

Fundamentals & Related Work

This Chapter introduces the basic concepts of mobility, cryptography and security protocols. A reference scenario is presented which deals with the mobility in IP networks. The challenges of providing seamless connectivity while roaming are discussed. Seamless connectivity in the Internet is facing a variety of obstacles especially when moving from one administrative domain to another, which is referred to as inter-domain handover. IETF approaches are described for IP mobility, MobileIP, as well as related performance enhancements in order to provide seamless connectivity. Furthermore, this chapter gives an overview on the current state of the art in cryptography, security properties, symmetric and asymmetric ciphers, important algorithms, digital signatures, keyed hashes and cryptographic protocols. Abstract protocols are discussed as well as real IETF proposals for securing Internet communication. The Internet security protocol suite (IPsec) together with its related key distribution mechanisms, and the Internet Key Exchange protocol (IKE) are also introduced in this chapter.

2.1 The Mobile Internet

Initially, the Internet was created as a robust, non-centralized network. It was not the purpose of the Internet to provide mobility and security for wireless connected nodes. Nowadays, wireless Internet-participants wish to stay seamlessly connected to the network even when moving across subdomains. This desire leads to enhancements that involve the provision of specific solutions for IP mobility. This is sometimes referred to as wireless IP. IP mobility in combination with security-guarantees enables cost-effective, high-quality IP-based wireless multimedia services, including that of *Voice over IP (VoIP)* [HCW01, KKS01, RS98, RS99], for a huge

Internet user community. The same trend can also be seen in multiplexing voice and data over the same packed switched network in the mobile phone world. The evolution from the *System for Mobile Communications (GSM)* [LHY99] cellular phones that belong to the *second generation (2G)* via *Universal Mobile Telecommunications Systems (UMTS)* [BA02] networks which are referred to as the *third generation (3G)* [Kor01] is assumed to end up with an all IP based *forth generation (4G)* [Bri01]. Mobility in 4G enables users to roam between different access-technologies such as satellite, UMTS and *Wireless Local Area Networks (WLANs)* [Rec04]. Mobility will be possible between the different service providers and sub-networks. This is also likely even among the fixed networks, hotspots and ad hoc networks. Mobility is achieved on the IP layer by extending the capability of the IP protocol, independently of the access technology.

2.1.1 Quality of Service and Resource Management

The Internet was originally designed as a connectionless best-effort network without any *Quality of Service (QoS)* [APF01] guarantees and with the vision of keeping the environment and protocols simple, robust, scalable and self-configuring. Real time and multimedia applications demand users to obtain adequate QoS guarantees. The aim of QoS support is to enable services to prioritize the transport of certain IP packets at the expense of packets carrying best effort traffic. Thus, stringent requirements have been put in place for premium data: data delay, delay jitter and packet loss.

Integrated Services (IntServ) [Arm00, BCS94], *Differentiated Services (DiffServ)* [BBC+98] and *Multiprotocol Label Switching (MPLS)* [RVC01] are current QoS techniques. These approaches have problems to provide end-to-end QoS in dynamic, heterogeneous IP networks with MNs as they are not scalable enough. In particular, re-negotiation of QoS settings at new access networks requires fast setting re-establishment mechanisms, although wireless links to the access network are generally communication bottlenecks. These are due to the movement the user wishes to set up in order to obtain QoS guarantees between the mobile device and the core network gateway.

Consequently, the result of QoS guarantees is: the users' willingness to pay and in the service providers' ability to charge for services. This requires security mechanisms to enable both users and network providers to authenticate each other. Hence, security mechanisms form the basis for the authorization of the network access. Moreover, the establishment of accounting methods is required in order to charge for such services. The conjunction of all these mechanisms is commonly referred to as *Authentication, Authorization and Accounting (AAA)* infrastructure [RHK+02].

Furthermore, combining seamless mobility and the establishment of QoS settings together with security properties involves a vast variety of protocols [FHS+04]. Some protocols aim to support mobility, the encryption and/or authentication of data, transport credentials, the establishment of security associations and to configure transfer modes by sending signaling messages. Most of such protocols have been proposed by the *Internet Engineering Task Force (IETF)*.

2.1.2 The Handover Reference Scenario

The reference scenario for IP mobility shown in Figure 2 shows all of the protocol actors and their connections as given in a wireless IP based environment [Dix02]. The roles are as follows:

Mobile Node (MN): The MN is commonly a battery powered low CPU power device that is moving (roaming) between the different access points. A user usually aims to profit from some of the services offered by network service providers. A session is normally established between the user and another party. This might for instance be a voice call.

Correspondent Node (CN): The CN is the communication partner of the MN. Here it is presented by some node which is somehow connected to the Internet. The CN can of course also be a mobile device.

Home Network: The Home Network of an MN recognizes the MN and trusts it. Home Networks might be a company's network or a service provider's network. Pre-shared keys exist between the MN and its Home Network.

Home Agent (HA): The HA is an actual instance located at the Home Network and serves as the contact host when the MN is not connected to the home network. The HA is the initial contact when trying to reach the MN and thus it should always be aware of the current location of the MN.

Access Point (AP) & Access Router (AR): The reference scenario shows two access networks (*Domain A* and *Domain B*). Three APs and two ARs (*AR-A1* and *AR-A2*) are inhabiting *Domain A*; *AR-B1* is the AR of *Domain B*. APs are limited in order to provide link layer connectivity, whereas the ARs do the actual routing of the IP traffic based on the IP addresses. Thus, APs are transparent for the IP traffic.

Authentication Authorization and Accounting Server (AAA): The Home AAA server (HAAA) stores all the MN's credentials and provides the authentication information when the MN wants to access services in a foreign network. The HA utilizes the HAAA server to authenticate the MN. Each domain has a local AAA server which is also referred to as foreign AAA server (FAAA). The border of a domain is defined by the FAAA in which the AR[1] is connected with.

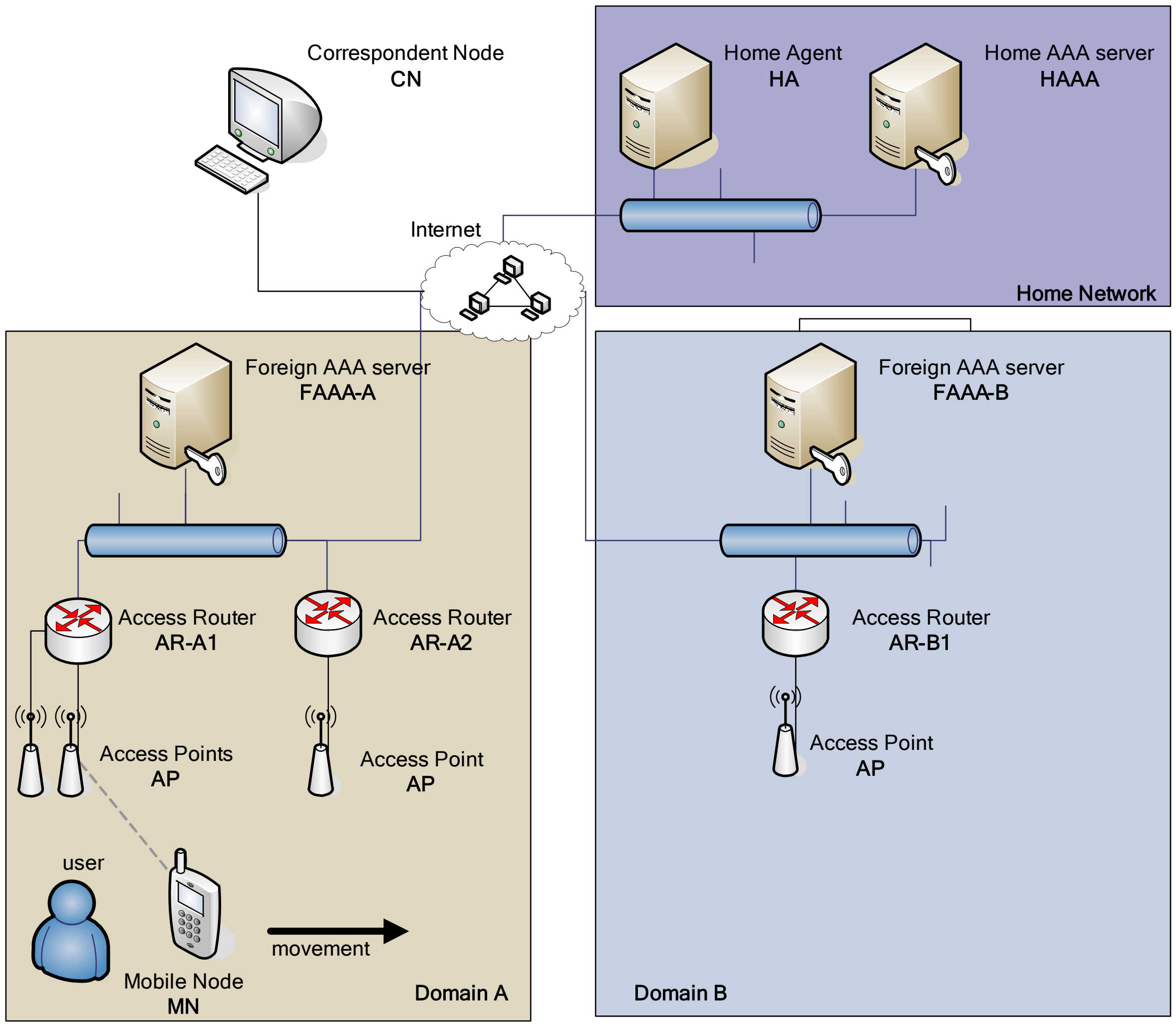


Figure 2: Mobility Reference Scenario

The active component in this scenario is the MN which is moving and changing the point of connectivity. This process of reconnecting to a new point of attachment is referred to as handover or handoff. A handover can occur on the link layer (layer 2). This means that an MN switches between two APs which are connected to the same AR. This so-called layer 2 handover has no influence on the network layer (layer 3)

[1] It is assumed that the ARs are only connected to one AAA server. It is further assumed that FAAA servers do not share any trust relationship or security association.

which is on top of the link layer. An MN, for example, moves within a wireless LAN environment (e.g. 802.11a, 802.11b, etc. [Rec04]) and switches from one WLAN access point to another. The IP address of the device does not need to change and so all IP packages can be routed as previously.The focus here is on the layer 3 handover where an MN moves from one AR to another. These IP layer handovers cause a change in the MN's IP address and therefore have an influence on the routing process. The link layer is also influenced by the layer 3 mobility and normally a layer 3 handover also implies a re-registration of the layer 2. Layer 2 advertisements are important for triggering the initiation of the handover. Handovers are either referred to as MN-initiated or network-initiated, depending on the entity that made the decision to perform the handover. Handovers are especially challenging for real time traffic as it is problematic to achieve seamless connectivity. Seamless means the combining of both a fast and a smooth handover.

Fast handovers aim to reduce handover latency and thus the delay caused by the re-establishing of the connection to the new AR. The MN is able to use the new link to send and receive packages again after it has been re-established.

Smooth handovers aim to reduce the amount of package loss (at best to zero) while performing a handover. Packets that arrive at the old point of attachment are usually dropped after the MN is disconnected.

Handovers between two ARs vary in the effort to fulfill the desired security and seamlessness requirements. As illustrated in the reference scenario in Figure 2, the MN performs a local handover while switching between *AR-1* and *AR-2* since they are both connected to the same network. A so-called ***intra-domain handover*** takes place whenever both ARs belong to the same administrative domain and therefore it is not necessary to change the access technology. It is much easier for most IP mobility improvements to be able to deal with these types of handovers. Security is also less complicated to achieve in the intra-domain case, since both ARs are managed by the same administration entity. Moreover, credentials are accordingly requested from the same server.

2.2 Mobile IP

Mobile IP (MIP) is an IETF proposal for providing macro-mobility in IP networks. In contrast to IPv4 where the Mobile IP version 4 (MIPv4) [Per02] is separated from the IP specification, with IPv6 the Mobile IP version 6 (MIPv6) [JPA04] is an integral part of the specification. The basic concept of MIPv4 and MIPv6 is described below.

2.2.1 Mobile IP Overview

The basic concept of Mobile IP is to have two IP addresses for each MN when located far from its home domain. This is due to the fact that the IP addresses have two different purposes. One is to uniquely identify the host, while the second is to enable routing. The former is a fixed so called home address that topologically belongs to the MN's home network, whereas the latter is required once the MN attaches itself to a foreign network. This so called *Care-of Address (CoA)* is used for routing purposes because classless inter-domain routing is based on agreeable subnet prefixes. Hence, the CoA changes whenever the MN moves to a new foreign domain. The CoA can either be obtained by stateful address auto configuration using the *Dynamic Host Configuration Protocol (DHCP)* [Dro02] or alternatively by stateless address auto configuration using the router advertisements. Routers typically broadcast advertisements at regular intervals. In cases when an MN needs to obtain a CoA and does not wish to wait for the periodic advertisement, the MN can broadcast or multicast a solicitation that will be answered by any router that receives it. Depending on the protocol used, the CoA either belongs to a so called *Foreign Agent (FA)* or to the MN itself. In the case of IPv4, the FA forwards packets to the MN directly on the link layer. However, with IPv6 the MN can have its own CoA and thus is able to receive packets directly.

A *Correspondent Node (CN*[2]*)* addresses packets to the MN's home address in order to begin communicating with it, as illustrated by (1) in Figure 3. When the MN is not connected to its home domain, packets are forwarded to a FA/CoA. The forwarding instance is a dedicated node, the HA, located at the MN's home domain. The HA as the tunnel entry encapsulates the entire IP packets and sends them to the FA/CoA (the tunnel exit) which in turn decapsulates the original packets. IP is commonly used either unencrypted/unauthenticated (IP-in-IP tunnel) or with encryption and/or authentication using IPsec (2.5.2). The tunnel between HA and FA is illustrated by (2) and the forwarding to the MN's layer 2 address is shown by (3) in Figure 3. Obviously, the HA always needs to be aware of the MN's CoA and hence the MN needs to register the new CoA in the home network. The relationship between the MN's home address and the CoA is called *binding*. Whenever obtaining a new CoA, the MN needs to send a binding update (e.g. an UDP Datagram). The MN registration request should be authenticated and the HA should approve the request before adding the CoA to its routing table. The association between the home address and the CoA is maintained until the registration lifetime expires. Packages

[2] The CN could also be a mobile node. For simplification, the CN is always represented by a fixed node.

are directly routed on their path back to the CN as illustrated by (4) in Figure 3. The routing between CN and MN is referred to as *triangular routing* due to its form.

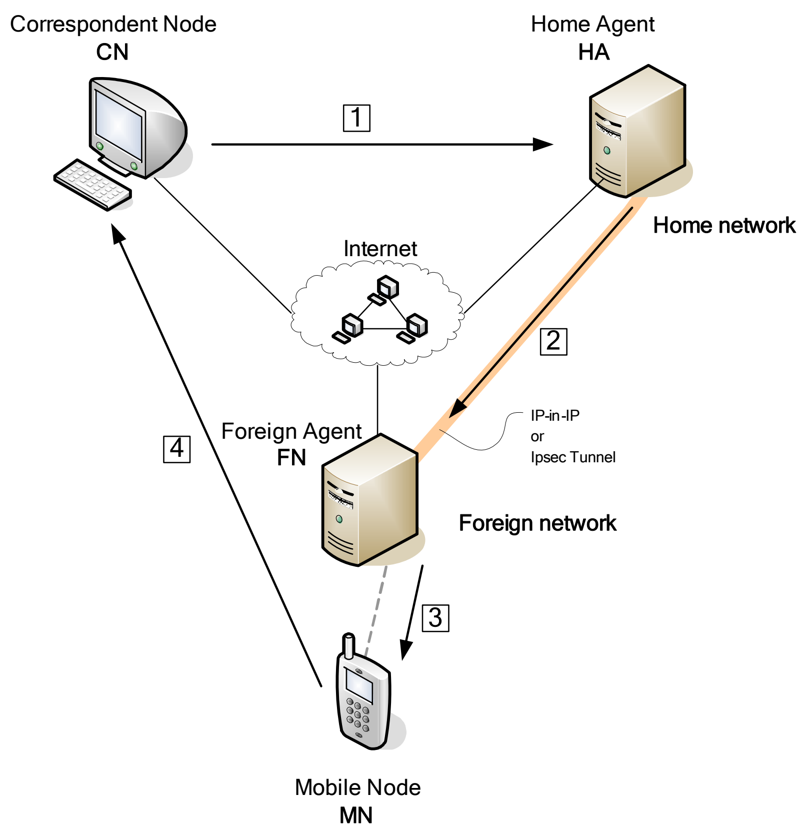


Figure 3: Mobile IP Routing

2.2.2 Mobile IP Extensions & Optimizations

Although Mobile IP provides host mobility, it is however facing numerous problems:

- Mobile IP uses topological non-correct IP addresses, that cause problems at firewalls due to ingress filtering;
- triangular routing creates performance problems;
- handovers may require overhands due to complicated binding updates and
- security for data and signaling needs to be provided.

Thus, a number of extensions and optimizations have been suggested in order to solve one or more of Mobile IP's shortcomings.

The extensions described herein endeavor to improve the performance in terms of reducing the handover latency. Some of these strongly require trust and security between access points when performing handovers. Hence, it is assumed that all of the necessary security requirements have been fulfilled. Internet security is considered in Section 2.5 .

2.2.2.1 Route Optimization

The task of route optimization is to avoid asymmetric triangular routing. One might imagine that the CN is not as distant from the MN as the HA is. This circumstance may result in an enormous extra latency. In the route optimized approach, the CA is informed about the MN's CoA and thus can send the packages directly to the MN [Sch04].

2.2.2.2 Reverse Tunneling

In contrast to route optimization, reverse tunneling sends and receives all traffic via the HA. The HA receives traffic from the MN and forwards it to the CN. The advantage of this approach is that this reverse tunnel prevents the problem of sending IP packages with incorrect sender IP addresses. In the normal Mobile IP approach, the MN uses its home address as the sender's ID. The problem here is the so called ingress filter which drops packages with incorrect sender IDs due to the *Denial-of-Service (DoS)* prevention.

2.2.2.3 Context Transfer

The *Context Transfer Protocol (CxTP)* denoted by RFC4067 [LNP+05] aims to enhance IP handover performance. When an MN moves to a new access network, it needs to continue certain transport-related services or services that have already been established at the previous subnet. Such services are called *context transfer candidate services*. Examples of these are states which are used in header compression, QoS reservations, AAA profile, IPsec, or firewall configuration [CB94]. Re-establishing these services at the new access network requires a considerable amount of time for the protocol exchanges. As a result, time-sensitive real-time traffic suffers. Alternatively, *context transfer data* can be forwarded, for example, from the *previous AR (pAR)* to the *new AR (nAR)* so that the services can be re-established quickly. It is one means of enabling the seamless IP handover operation of application streams and could possibly reduce the susceptibility to errors. Furthermore, service re-initiation to and from the MN can be avoided, thereby maintaining the wireless bandwidth efficiency. CxTP is assumed to be typically used for intra-domain handovers since it requires trust relationships between the ARs.

2.2.2.4 Cellular IP

The aim of cellular IP [Cam00] is to enhance the handover performance by providing so called micro mobility for a small compartment. Within this compartment, the MN is not involved in the authentication process. The geographical area is divided into small compartments called cells. Each cell is covered by a number of ARs that wholly communicate with a dedicated gateway router. The cellular IP network is connected to the IP backbone by means of the gateway router. The gateway router sends out a so called *beacon* to all of the ARs in order to keep the gateway advertised as a potential routing destination. Regardless of the destination address, all ARs route the packets directly to the gateway router.

In a cellular IP network this functions vice versa, where the MN uses the address of the gateway node as its current CoA. The gateway router in charge of the MN and the intermediate nodes in the cellular IP network, permanently need to be aware of the MN's current AR. Thus all nodes need to maintain two caches: a mandatory routing cache and an optional paging cache. The routing cache contains a mapping between the MN's ID and its corresponding router. This is the router from which the MN received the packet. The mapping is set when the MN sends a packet towards the gateway router[3]. The paging cache is similar to the routing cache but has a longer lifetime for the event where the MN cannot be found in the routing caches. Whenever an MN moves to a new location, it needs to send a route update packet in order to update cache entries in all intermediate nodes along the path from the AR to the gateway router.

Cellular IP only provides micro mobility and therefore does not need to send any binding update when the MN remains in the same cellular IP network. Whenever the MN moves to a new cell, a renewed Mobile IP update has to be produced in order to change the CoA binding in the home network. Messages are from now on routed either to a new gateway router or to a non cellular IP aware AR acting as FA. Therefore, cellular IP provides faster handover within the boundaries of the cellular IP network.

2.2.2.5 Hawaii Model

The Hawaii model [Ram00] is similar to the cellular IP in it also provides micro mobility. The architecture consists of cells where each cell contains a crossover router and ARs. In contrast to the cellular IP where the MN uses the gateway router's IP

[3] The entry remains valid for a limited period. In order to maintain the validity of the routing table, the MN needs to send data packets or update packets (ICMP route-update-packets).

address as CoA, the MN acquires a dynamic IP address from the DHCP server of the foreign network.

When the MN moves to a new domain within the same cell, it will register itself with the new AR by sending an update to the crossover router. However, in the event that the MN switches to a different cell, it needs to reproduce the normal Mobile IP binding update.

2.2.2.6 Hierarchical Mobile IP

The third approach for providing micro mobility in geographic cells is *Hierarchical Mobile IP (HMIP)*. Unlike the cellular IP and the Hawaii model, the HMIP organizes nodes within a tree type cell which is the reason why it is called hierarchical. The gateway router is the root and ARs are the leaves. However, not all nodes in the HMIP are involved in the update procedure, as is the case in the other micro mobility approaches. In HMIP, the MN only informs the nodes up to the point where the hierarchy is affected. A *Mobility Anchor Point (MAP)* is an optional management point providing a regional CoA. The MAP can be located at any level of the hierarchy between the AR and the gateway router.

2.3 Cryptography and Security

Seamless connectivity over non-centralized IP based networks raises questions on security considerations. The U.S. National Information Systems Security Glossary (INFOSEC) defines information systems security as:

"the protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit and against the denial of service to authorized users or the provision of service to unauthorized users, including those measures necessary to detect, document and counter such threats".

Protocols aiming to provide security in heterogenic environments need to fulfill certain security properties based on mathematical functions. A deep understanding of all the mathematical functions is not required in order to comprehend security protocols. However, protocol designers should be aware that these functions provide a toolset for cryptography [KPS02, SH04].

The purpose of cryptography is to exchange information between two or more parties in a way that no unauthorized third party can obtain the information. Thus, a plain text message is transformed by a mathematical function which rearranges the output into a random string. Although this random code can be viewed by all persons, only those who are principals are able read the original message. The transformation of plaintext to cyphertext is called encryption and the complementary operation of converting cyphertext to plaintext is called decryption.

2.3.1 Security Services & Properties

When designing a protocol for a given environment it is important to advisedly define all properties and services that the protocol should provide. Based on cryptographic algorithms, the tasks of security services are to provide at least one of the following properties:

Data Integrity guarantees that the data received is the same as that which has been sent or at least that any changes will be noticed by the receiver, even if the original data can not be reconstructed. Integrity is especially required when data is sent over insecure channels. Checksums such as the *Cyclic Redundancy Check (CRC)* and attached hashes prevent bit failures caused by transmission errors, whereas keyed hashes and *Message Authentication Codes (MACs)* [Tsu92] provide confidence that messages have not been deliberately modified.

Data Confidentiality (Secrecy) ensures that information is only accessible for authorized entities and that no information is provided to anybody else. Encryption is used so that data can be extracted by only those who know the required key. Encryption and decryption are the essential ingredients for cryptographic protocols in order to ensure data confidentiality. Therefore, the common encrypting mechanisms are explained in detail later in this chapter.

Data Origin Authentication guarantees that the origin actually matches the claimed identifier. Data origin authentication can be considered as integrity over the binding of data with the corresponding sender's ID. Thus, data origin authentication implies data integrity. Data origin authentication is fundamental for entity and user authentication and is the basis of all session-key establishment mechanisms. Therefore, data origin authentication is often used as an equivalent to integrity, even if integrity does not necessarily imply user or entity authentication. Experts agree [Gol01] that it does not make any sense to have data integrity without authentication. It should be mentioned that in common security notation the sender and receiver IDs are not part of the message. However, if integrity is assumed, this also applies to the sender's ID.

Entity- & User-Authentication provides proof as to whether the sender of a message (entity or user) is actually the person he or she claims to be [Gol96]. No person other than the sender is able to authenticate the information. There is no distinction made in these papers between the user and entity authentication. Thus, ***Authentication***

refers to either user or entity authentication and involves data origin authentication as being a fundamental part.

Authorization & Access Control are mechanisms for deciding whether a communication party is allowed to access information or can make use of services that are only provided for specific users. This is based on common user authentication and enables the matching of the entity's data with the preconfigured properties stored at a specific database. However, authorization does not necessarily imply authentication as anonymous paying systems such as electronic coins can be based on authorization only.

Non-Repudiation is a service to bind the sender's identity to the data in order to ensure that users cannot deny having sent the data and is thus the domain of digitally signing messages. This is especially important for legal or billing purposes. Non-repudiation implies data integrity and data origin authentication, but does not imply any confidentiality.

Anonymity/Privacy are properties that protect the user's identity. An observer should be unable to determine which event occurred when or from where it had originated. Obviously, this definition depends on the standpoint of the formal observer. A possible solution to ensure anonymity of messages is modeling a system by renaming the origin of each message in a nondeterministic manner.

The above mentioned properties are the essential requirements for a secure information exchange in modern networks. These requirements are fulfilled by means of cryptographic algorithms which are presented in subsequent sections. The algorithms introduced here are grouped as symmetric cryptography, asymmetric cryptography, authentication mechanism and key exchange protocols.

2.3.2 Symmetric Cryptography

The basis of symmetric cryptography is a shared key *K* which is only known to the communicating parties. The mathematical function is symmetric in the sense that the same key is used for both encryption and decryption. The encryption algorithm *F* and the decryption algorithm *F'* are assumed to be well known so that security only depends on the knowledge of the key. The ciphermessage *C* is computed from the plaintext *P* as $C = F(P,K)$. The receiver of the message derives the Plaintext from *C* as $P = F'(C,K)$. The term $\{X\}_K$ expresses that *X* is encrypted by key *K* while *X* can be any data either already encrypted or plaintext. Nested brackets refer to multiple encryptions. For instance, a plaintext *P* encrypted by *K1* and then further encrypted by *K2* is expressed by $\{\{P\}_{K1}\}_{K2}$. Appendix A introduces the common security protocol syntax as a context free grammar.

The main advantage of symmetric cryptography is that the algorithms *F* and *F′* are easy to implement. Furthermore, high speed is achieved in the process of encrypting and decrypting messages. Symmetric cryptography can even be very efficiently realized in hardware which makes it feasible for the use of low CPU power devices such as set top boxes and mobile devices.

However, a serious disadvantage in using symmetric cryptography is due to the requirement of having to have a shared key. This leads to the problem of the key distribution in cases where the two parties have not already exchanged a pre-shared key. A further complication is that each key is only valid for two communication entities and therefore it increases the complexity of storing keys for all potential communicating partners. One possible solution is to use a secure key establishment mechanism prior to encryption. Meanwhile, this problem can be solved by asymmetric cryptography that is based on a public/private key pair rather than on a single private key.

Symmetric ciphers that are used nowadays can be distinguished between two groups, namely block and stream ciphers. Block ciphers [Lai92] require the message to be split into blocks of a fixed length. Encryption and decryption can easily be implemented, especially as a software solution. The last part of the message after splitting it is most likely to be smaller than the fixed length and therefore a so called padding is attached to expand the part to the required size. This padding is a well defined attachment of the message that is removed when the block is received. Stream ciphers are usually developed as a hardware solution and they are usually faster to compute than block ciphers. One possible solution for stream ciphers is to create a pseudo random stream. This stream is generated by the dependence on a key and a start value which is referred to as *Initialization Vector (IV)*. This pseudo random stream is used to perform an *Exclusive OR (XOR)* bit-by-bit. The concept is based on the fact that an XOR operation with a real random stream is only used once. This is called *One Time Pad (OTP)* and is provably secure if the random stream is not known by anybody except the communicating parties. However, the security depends on the quality of the pseudo random string and many of the systems available today are considered to be insecure for this reason.

The *Data Encryption Standard (DES)* [NBS77] is an example of a 64-bit block cipher which came to be a widely spread algorithm for symmetric cryptography. DES is cryptographically strong since the best attack known is the brute force attack. This means the only way to obtain the key is to attempt all possibilities. The key length is 56 bits plus 8 bits for parity checking. However, this cipher is not considered to be secure any more due to the key length of 56 bits being too short. Nevertheless, it was sufficient 20 years ago when the DES was being developed.

Hence, several successors of the DES have been proposed in order to improve the key length. The *Advanced Encryption Standard (AES)* [DR02] for instance, was introduced in 1997 and can operate with different block and key sizes. The block sizes are usually128 bits and the possible key lengths are 128, 192 and 256 bits. Table 1 gives an overview of some commonly used ciphers.

Category	**Type**	**Example Algorithms**	**Service**
Symmetric	Streamcipher	RC2 [Riv98] RC4 [Riv92b] RC5 [Riv95, Riv95a] SEAL [RC94] WAKE [Whe94]	Confidentiality
	Blockcipher	DES [NGK77] AESIDEA SAFER [Mas94, Mas95] Blowfish [Sch94, Sch94a]	
	MAC	HMAC [KBC97]	Data Integrity
	Hash Function	SHA-1 [NIS94] MD4 [Riv92] MD5 [Riv92a] HAVAL [ZPS93]	One-Way-Function
Asymmetric	Cipher	ElGamal [Gam85, Gam85a] Cramer-Shoup, RSA-OAEP	Confidentiality
	Digital Signature	RSA Signature [RSA78]	Non-Repudiation

Table 1: Examples for Commonly Used Cryptographic Algorithms

2.3.3 Asymmetric Cryptography

In contrast to symmetric cryptography where a single secret key is shared, asymmetric cryptography operates with two different keys for each party. One key is referred to as the *private key* since it is only know by its initiator and is not shared with anybody else. Each private key has a corresponding *public key* that is distributed and should be at best publicly available. Therefore, asymmetric cryptography is also called *public key cryptography*, while symmetric cryptography is also referred to as *secret key cryptography*.

The concept of asymmetric cryptography is that a plaintext P is converted into a cyphertext C by means of the public key K_{pub}, $C = F\left(P, K_{pub}\right)$. The original message can only be derived from the cyphertext by using the corresponding private key K_{priv}, $P = F^{-1}\left(C, K_{priv}\right)$. It has got to be guaranteed that it is impossible to gain knowledge of the private key when one possesses the public key. Hence, security depends on the knowledge of the private key.

As each key pair belongs to one entity, the key name includes the entity's name. Hence, the public and private key of an entity A is expressed as: *PK(A)* for the public key and *SK(A)* for the secret key. The "secret key" also refers to the private key which is called "secret" due to the abbreviation "SK", whereas "PK" refers to public key. Messages that contain X encrypted by the public key of A are expressed as $\{X\}_{PK(A)}$.

The obvious advantage of asymmetric ciphers in comparison to symmetric ones, is that no secret information has to be exchanged over probably insecure channels and everyone only needs to possess a single private key for decryption. However, there are some serious disadvantages as well. The involved mathematical operations are far more complex in so much that encryption and decryption require more computation effort and are consequently slower than the symmetric approach. A further disadvantage is that the keys for asymmetric cryptography need to be larger in order to achieve a similar level of security. A 128-bit symmetric key for instance, acquires the same level of security as a 3000-bit asymmetric key.

The most famous asymmetric algorithm is *RSA* [RSA78, ACG+84, ACG+88] named after its inventors R. Rivest, A. Shamir and L. Adleman who developed it at the *Massachusetts Institute of Technology (MIT)* in 1977. The strength of the algorithm is based on the actuality that factoring large numbers is difficult [Adl91, Riv93]. More details on the mathematical background into asymmetric ciphers can be found in [BR95]. The RSA patent expired in 2000, thus enabling the algorithm to be used freely nowadays.

2.3.4 Symmetric vs. Asymmetric Cryptography

Comparisons of the advantages and disadvantages of symmetric and asymmetric cryptography lead to a hybrid approach that combines both mechanisms. As the symmetric approach provides higher performance and is easier to implement in hardware it is used for data encryption whereas the symmetric key is exchanged based on asymmetric cryptography [BCY92].

A further application for asymmetric ciphers is to use them for signing messages. Therefore, plaintext is decrypted by using the sender's secret key and is transmitted by attaching it to the plaintext. As the sender's public key should be publicly available and the mechanism also works in the inverse way, everyone should be able to decrypt the message by using the sender's public key. A digital signature of A signing plaintext X is expressed as expressed as$\{X\}_{SK(A)}$.Hence, symmetric cryptography in conjunction asymmetric cryptography provide:

Authentication: Once the receiver succeeds in decrypting the information with the sender's public key, he can be sure that the plaintext has actually been encrypted by the sender. One problem here is that it is necessary to prove that the claimed public key actually belongs to the sender. This can be solved by means of digital certificates and a *Public Key Infrastructure (PKI)*, which is explained in more detail in Section 2.3.5.

Integrity is also provided for digitally signing, as a decrypted cipher message and plaintext are not equal if either cipher or plaintext has been modified in transit.

Non-repudiation is provided because only the sender could have transmitted the message. No other person could have generated the cipher text without possessing the private key. Therefore, the sender cannot deny having sent the message. A timestamp is usually included in the message in order to ensure that the message is fresh and that no attacker has been able to resend it.

2.3.5 Public Key Infrastructure

A *Public Key Infrastructure (PKI)* [BHR99] is an agreement between trusted third-parties that vouch for user identities. These third parties are servers that vet and distribute signed certificates. Basically, a certificate binds a user's public key together with the corresponding user's identity. Certificates are digitally signed by a trusted third party. Thus, users trusting a common third party's public key will also have trust in each other's public keys.

The PKI certificates [HFP+99] are the basis for authenticating users with each other. Hence, by digitally signing messages using one's private key, other users are able to verify the validity of the signatures. Therefore, in order to achieve this, they use the corresponding public key that is contained in the user's certificate.

2.3.6 Hash Functions, MACs and One Way Functions

Signatures as provided by asymmetric cryptography would consume too many resources if applied to the entire document. Thus, a so called *hash function* is used to compile small fixed length outputs (e.g., 128 or 160 bits) of an arbitrarily variable input size. The output is called *message digest (MD)* and is further signed by using the sender's private key. This function is referred to as *one way function* [Gon89, Sch91, BM97], as it is computationally difficult to retrieve the original data from a digest. Further, minor changes to the original input-data should result in totally different output-digests.

Hashes need to be resistant against brute force and *birthday attacks* [Gon95]. A birthday attack is an approach in order to obtain two messages that have equal digest-outputs. This output is retrieved by altering variable elements of both messages. Each variable explores the scope and thus the probability is unexpectedly high in actually receiving two messages with the same digest. Therefore, hashes consisting of a few hundred bits are required. A 64-bit digest would for instance, be vulnerable against birthday attacks, since the probability of having two equal messages exceeds 50% by just increasing the amount of messages to 232.

Hash functions that are widely used are: MD2 [Kal92], MD4 [Riv90, Riv91, Riv92], MD5 [Riv92a], SHA-1 [MG98a, NIS94], RIPEMD-160. In addition to normal hashes one could use keyed hashes that are called *Message Authentication Codes (MACs)* [BCK96]. MACs aim to provide integrity and data origin authentication with the aid of a shared secret key. The *Hashed Message Authentication Code (HMAC)* [KBC97] is a specific MAC algorithm consisting of two components: firstly, a cryptographic hash function and secondly, a secret key as an input parameter.

Hash functions themselves are not suitable for authentication use purposes, as they do not share any secret key between sender and receiver. HMAC algorithms combine hash functions that basically do not provide any encryption or authentication with a secret shared key. Therefore, HMACs can be used for authenticating communication endpoints. Popular examples are HMAC-MD5 [MG98] and HMAC-SHA1 [MG98a]. Both of them can be used as IP message authentication within IPsec as described in Section 2.5.2.

2.4 Cryptographic Protocols

Protocols are abstract conventions or concrete standards that control or enable connections, communications and data transfers between entities. Telecommunication is based on *communication protocols* which describe the interaction rules for sending information over a communication channel. They are represented by a set of rules specifying data representation, signaling, authentication and error detection.

Cryptographic algorithms, as stated above, provide cryptographic services on data such as confidentiality by encryption and non-repudiation as well as integrity by digitally signing and cryptographic hash functions. The combination of communication protocols and cryptographic algorithms form the basis for *cryptographic protocols* (or security protocols). They rely on transport mechanisms provided by communication protocols and utilize services of cryptographic algorithms in order to accomplish specific services between communicating entities. Cryptographic protocols commonly incorporate at least some of the following aspects:

- Key agreement, negotiation, or establishment [BM00]
- Entity or user authentication
- Symmetric encryption and data authentication
- Secured application-level data transport
- Non-repudiation methods

The tasks of cryptographic protocols vary from offering essential cryptographic services to complex responsibilities. Depending on their purpose, cryptographic protocols can be distinguished as follows:

1. ***Essential cryptographic protocols***: These protocols simply provide cryptographic properties between the communication entities that are directly offered by the corresponding cryptographic algorithm. Data, for instance, is encrypted at the sender-site, transferred and decrypted at the receiver. Thus, such protocols provide data confidentiality by en- and decryption. These protocols utilize algorithms together with keys to compute cipher messages. They may either use pre-shared keys or public private key pairs. When a pre-shared key is used, data is also authenticated as only one of the key possessors could have sent the data. These protocols either authenticate messages by encryption or by attaching digital signatures that may be based on hashes. Some protocols imply a specific algorithm, whereas others are flexible in offering and negotiating cryptographic algorithms which are supported at both communication endpoints. However, the characteristic that all of these protocols have in common, is they assume that the key has already been established prior to the protocol-start. They further assume that the communicating entities have been authenticated and thus deal with data

authentication and integrity, but neither with the user- nor with the entity-authentication.

2. ***Authentication and Session-Key Establishment Protocols***: This group of protocols also utilizes cryptographic services, but provide more complex properties. Hence, these protocols employ encryption algorithms, hashes and digital signatures as essential services. A major purpose of these complex protocols is to provide authentication between the communication entities. Some of them involve a trusted third party, while others are server-less. Authentication protocols commonly establish a session-key that is shared by both entities and is valid for only a limited amount of time. The purpose of the session-key is for it to be used by further cryptographic protocols in providing authentication and confidentiality for messages in transit. Moreover, some protocols exist that provide authentication without any session-key establishment. These are somewhat controversial, as they lack any practical relevance. Nevertheless, session-key establishment does not imply authentication. Some protocols establish session-keys even without authenticating them.

The two groups of cryptographic protocols interoperate with each other, since the second group establishes such session-keys which are then used by the first group. Thus, the security of the entire system is dependent on the authentication procedure and the quality of the session-key. Authentication and session-key establishment is considered in more detail in the following section.

2.4.1 Authentication and Session-Key Establishment

Authentication and key session establishment are fundamental for securing communication protocols. Cryptographic algorithms providing data confidentiality and integrity, can only perform their function if secure keys have been established and the participating entities are aware of which parties share such keys. Authentication and session-key establishment protocols are used for this purpose.

Cryptographic protocols are classified according to their influencing criteria. These criteria are preconditions for all principals, key generation method requirements, the number of involved participants and their roles.

Preconditions describe which keys have already been established at which entity. This includes the awareness of other entities, the knowledge of public keys and the trust between entities. It is a matter of fact, that it is impossible to establish an authenticated session without prior existence of either shared secrets or certified public keys between the entities (this has been asserted and proven to be true [Boy93]). Therefore, entities either already share a secret key, are capable of utilizing an offline server for authentication (this is also referred to as verifying claimed public keys), or they have access to a trusted third party who acts as the authentication server.

The ***key generation method*** distinguishes protocols according to the principal that actually generates the session-key or principals involved in the process. A *key transport protocol* is a method where one entity generates the session-key, commonly by a random function and then distributes it to other principals. A *key agreement protocol* is a method in which the key is generated by a function. This includes input parameters for all further users of the session-key. A *hybrid protocol* generates session-keys based on inputs of more than one principal, but not of all users. This can be the case, if a trusted server and one dedicated user, influence and thus agree on the session-key. Other users would view this protocol as merely being a simple key transport.

The ***number and role of involved principals*** describe the actions of principals involving at least two, but theoretically unlimited numbers. Principals are considered as *servers,* if they do not intend utilizing the session-key for further communications. Whereas *users* are principals who potentially make use of the session-key for further security mechanisms.

2.4.1.1 Goals of Authentication and Key Establishment Mechanisms

The purpose of the security protocol run can vary depending on the desired goal of each principal. The fact that an entity A wants to communicate securely with another entity B based on a session-key K, implies that A needs assurance that only B knows the key (secrecy of K) and that B is actually the desired entity (authentication of B). Authentication generally describes the process of A determining whether another party B is in fact the party it claims to be. In the event that B also wishes to communicate with A in a secure manner, this is termed mutual authentication. Thus, the objective of authentication and session-key establishment protocols is to authenticate entities and to guarantee confidentiality for the key. Furthermore, keys are required to be fresh. This denotes that the key has never been used or transferred before a certain point in time.

Authentication is achieved by agreeing on variables that can also be identifiers. Each agreement and secrecy statement is a basic aim. Security protocols commonly aim to fulfill several secrecy and agreement statements. These statements are formally defined as:

Secrecy statements specify the objective that confidential data is only known to desired entities. A secrecy statement formally specifies the protection of confidential data against an intruder:

$$Secret(A,K,[B])$$

The statement above defines the assurance of A concerning confidential data K to be known only to him and optionally to B. This statement expresses that any intruder does not have any possibility of acquiring possession of the secret. However, this does not claim any assurance that the data is actually known to B. Moreover, protocols involving a trusted third party tend to accept that the server may also have knowledge of the secret.

Agreement statements specify the aim that one entity A is assured of the identity of a second entity B. Authentication can be formally considered as an agreement on a variable. The agreement statement can be expressed as follows:

$$Agreement(A,B,[V_A,V_B])$$

This statement is interpreted as the assurance. Therefore, case A completes a run of the protocol, apparently with B, then B has run the protocol, apparently with A; consequently, both principals have agreed upon each other's roles and upon the values of V_A (involved by A) and of V_B (involved by B). There is a one-to-one relationship between the runs of A and those of B.

It is not necessary that V_A or V_B have secrets between A and B. Hence, if A and B wishes to agree on a secret key, a supplementary secrecy statement is required. However, in the event that V_A and V_B have never been previously used, freshness is guaranteed.

2.4.1.2 The Freshness Property of a Session-Key

In contrast to long term keys, a session-key is valid only for a limited amount of time. This period is referred to as a session. The idea is to generate and distribute session-keys on demand and to delete them after the session has ended. Apart from the user oriented objectives for secrecy and authentication, an additional key oriented aim is that of freshness. The concept of freshness denotes the up-to-dateness of security data such as keys and messages. Keys and variables are referred to as being fresh if they have never been used before and at best, have momentarily been generated prior to usage. Thus, a *good key* for A to use with B is defined as a key that is only known to A and B (with the exception of some trusted third party) and is moreover a fresh key. Freshness of a key can be guaranteed by deriving the actual key from a function that has, other than a long term secret, a fresh variable as input. This variable is referred to as a nonce.

Nonces

A nonce, in the context of security protocols, is a (pseudo-)random value generated by a pseudo random generator at one principal. The space of possible values is required to be sufficiently large (e.g. at least 128 bits) and the generation process must produce unpredictable output-values.

Nonces are used in cryptographic protocols in order to guarantee freshness during the protocol run by establishing causal relationships between the messages. Principals involving a nonce in the protocol are subsequently able to verify the freshness of responses correlating to messages sent previously. A common mechanism known as *Challenge Response Mechanism* aims to authenticate participants by also guaranteeing freshness of the authentication. Assuming *A* wants to authenticate *B*, a challenge response mechanism works as follows:

1. It is assumed that *A* and *B* share a long term secret *S* and both know a common one way function *F*.
2. *A* generates a nonce *na* at a certain point in time *t* by utilizing a random generator. This nonce is also considered as a challenge.
3. *A* sends the nonce *na* to *B*.
4. *B* while receiving *na* computes a response *r* by the well known function *F* so that the response *r := F(na, S)* is dependent on the secret *S* and the challenge *na*.
5. *B* sends *r* as a reply to *A*.
6. *A* also computes *F(na, S)* and compares function output with the received *r*. When these values are equal *A* assumes *B* as authenticated. The authentication is fresh since *B* has to have computed *r* after a known point in time *t*.

It should be mentioned here that the nonce *na* is sent in plaintext and thus can be read by anyone. Counters or timestamps can also be used to provide freshness in lieu of random values.

Timestamps

While nonces are only local markers in time, meaningful to the creator only, timestamps act as global markers and are meaningful to every principal. Timestamps are especially important when protocols should provide freshness to more that one participant. Furthermore, the number of messages is reduced. However, random nonces require a challenge exchange for each principal in order to authenticate others, whereas encrypted identifiers and timestamps involve a single message for authentication. Therefore, timestamps have significant drawbacks. Clocks are required at each principal and time needs to be synchronized between all participants and at best even globally. Validity checking and interpretation of timestamps is also more expensive in terms of computation power than forwarding or modifying nonces.

2.4.1.3 Forward Secrecy

Forward secrecy, also referred to as *Perfect Forward Secrecy (PFS)*, is an extended goal which is provided by session-keys. Forward secrecy aims to provide confidentiality after one session has ended, even if the participants are then compromised.

In cases where a session-key is sent from one principal to another via a secured channel, the key is also considered secure. An intruder that has meanwhile captured the entire communication is capable of decrypting the secured channel once it has compromised at least one of the agents. The intruder is then able to decrypt the secured channel in order to obtain the session-key which was employed for encrypting the entire captured communication.

Therefore, forward secrecy is a property that prevents intruders from obtaining any data on the condition that once a session has ended, both agents delete their session-keys and copies of the confidential data are no longer stored by any principal. Forward secrecy can be achieved by utilizing the mathematical concept of one way functions while each agent constructs a so called *half key*.

Assume the two one-way functions *F1* and *F2* (both can even be publicly known) and also take into consideration the following equivalence:

$$\mathit{Session\ Key} := F2\big(F1(a),b\big) = F2\big(F1(b),a\big)$$

The session-key is established as follows:

1. *A* chooses *a* while *B* chooses *b* respectively.
2. *A* send *F1(a)* to *B* whereas *B* sends *F1(b)* to *A.*
3. Both *A* and *B* compute the session-key due to the above mentioned equivalence.

Trivially, when knowing *a* and *b* one can compute the session-key. Hence, once the session-key is generated *A* and *B* delete *a* and *b*. It has also got to be guaranteed that no function *F3* exists (or at least is not efficiently computable) to compute the session-key based on both half keys:

$$\mathit{Session\ Key} = F3\big(F1(a),F2(b)\big)$$

In Section 2.4.2.2, the Diffie-Hellman Key Exchange Protocol is introduced as an abstract protocol providing forward secrecy.

2.4.2 Authentication Protocol Approaches

Two different approaches exist for the authentication of protocols. The first mechanism, called *arbitrated authentication protocol,* relies on a trusted third party. The second method does not need any direct access to a third party for providing authentication between the two entities and is therefore called *direct authentication.*

2.4.2.1 Authentication by a Third Party - The Needham-Schroeder Example

Authentication between two parties *A* and *B* can be achieved by a third party trusted by both *A* and *B*. Each time *A* and *B* desire to authenticate each other it involves a unique trusted party *C*. Many different protocols have been developed for this purpose. It depends on the various conditions that apply to the party requiring authentication. This could even be on a mutual basis. A simple protocol for this purpose is the *Needham-Schroeder shared Key Protocol* which is based on symmetric encryption between the communicating parties. The trusted party *C* shares a key with each party. Two keys exist in this simple example, one shared between *A* and *C* referred to as K_{AC} *and* one shared between *B* and *C* termed K_{BC}. The purpose of the authentication procedure is to establish the shared key K_{AB} between *A* and *B*. Assuming that *A* wants to talk to *B, A* begins by sending a message to *C*. This message is not encrypted and includes the following parameters:

- *IDA*: Identification of *A*
- *IDB*: Identification of *B*
- A_{ran}: A random number created by *A*

When *C* receives the message from *A* it generates a session-key K_{AB} for any subsequent communication between *A* and *B*. *C* then sends a message back to *A*:

$$\left\{A_{ran},\ IDB,\ K_{AB},\ \left\{IDA,\ K_{AB}\right\}_{K_{BC}}\right\}_{K_{AC}}$$

The message is totally encrypted by the key shared between *A* and *C,* so that only *A* is able to decrypt the message. Firstly, *A* checks if the random number is the one it used for the request. This guarantees that the message is fresh and that the answer received is not a replay attack by some malicious sender. The message contains the ID of *B* so that *A* knows what session-key it has received; one might also consider the case of *A* requesting more than one key at the same time.

The session-key K_{AB} is actually included twice in the message, but has only been encrypted once by K_{AC} .It has also been bound together by *A*'s ID and encrypted by the key shared between *B* and *C*. *A* forwards the part encrypted by K_{BC} to *B*. *B* then generates a random number B_{ran} and sends $\{B_{ran}\}_{K_{AB}}$, the encrypted random number back to *A* by means of the shared session-key. When the message is received, *A* is able to decrypt it since it also has the shared key K_{AB}. *A* subsequently decrements the random number B_{ran} and encrypts it by means of K_{AB}. Thus, *A* sends $\{B_{ran}\text{-}1\}_{K_{AB}}$ back to *B*.

This simple protocol illustrates session-key establishment and authentication. Initially, both parties receive the session-key from *C*, directly in the case of *A* and indirectly in the case of *B*. This element pertains to the key establishment mechanism. The ultimate two messages in which *A* receives and returns the random number, correspond to the authentication part of the protocol. This aims to prove the identity of *A*. Authentication here denotes proving that *A* has in fact knowledge of the key K_{AB}. This indicates that *A* shares a key with the trusted party *C* and is thus considered trustworthy.

2.4.2.2 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange (DH), initially published in 1976 [DH76], is a cryptographic protocol which allows two principals having no prior knowledge of each other to establish a session-key over an insecure communication-channel. This authentication mechanism does not involve any third party and is thus called server-less or direct authentication.

Both parties *A* and *B* agree on a finite cyclic group *G* and a generating element *g* in *G*. *G* is a multiplicative group modulo *p* while p is relative prime in *G* [APR83]. Integers between 1 and $p-1$ are used with normal multiplication, exponentiation and division, except for the fact that after each operation the result retains only the remainder after it is divided by *p*.

The distribution of *G*, *p* and *g* is commonly carried out long before the rest of the protocol. *G*, *p* and *g* are assumed to be known by everyone including all potential attackers. The HD protocol works as follows:

1. *A* chooses a random natural number *a* and sends $g^a \bmod p$ to *B*.
2. *B* also selects a random natural number *b* and sends $g^b \bmod p$ to *A* respectively.
3. *A* computes the session-key as $K_a := \left(g^b \bmod p\right)^a \bmod p$.
4. *B* computes the session-key as $K_b := \left(g^a \bmod p\right)^b \bmod p$.

Finally, both *A* and *B* share a session-key *K* due to the homomorphic property of exponentiation:

$$K := K_a := \left(g^b \bmod p\right)^a \bmod p = g^{b+a} \bmod p = g^{a+b} \bmod p = \left(g^a \bmod p\right)^b \bmod p =: K_b$$

Although DH lacks any authentication, it provides the basis for a vast range of authentication protocols. It should be mentioned that DH provides forward secrecy for the session-key *K*, as the subsequent deletion of *a*, *b* and *K* at both parties, does not allow for any previous communication to be reconstructed, even when compromising *A* and *B*. Consequently, DH is considered secure against eavesdroppers if *G*, *p* and *g* are properly selected. Deriving *K* from given $g^a \bmod p$ and $g^b \bmod p$ is currently considered difficult, as there is no known efficient algorithm available for solving the discrete logarithm problem [Adl79]. However, since DH by itself does not authenticate messages, it is vulnerable to man-in-the-middle-attacks (see Section 2.4.3). Hence, DH is commonly utilized for establishing a session-key via a secured channel which implies authentication. DH then provides forward secrecy for a secured and authenticated session-key [BKP00].

2.4.3 Protocol Vulnerabilities

Communication protocols should not end up in deadlocks or livelocks due to misbehaving nodes, message losses or incorrect messages caused by bit failures in transit. Nevertheless, security protocols are facing additional vulnerabilities from malicious nodes that behave mischievously on purpose. These nodes are referred to as attackers or intruders that aim to break the protocol for a vast variety of reasons. Security protocols have to be designed to deal with attacks that are more or less present and are dependent on the considered environment and the given assumptions.

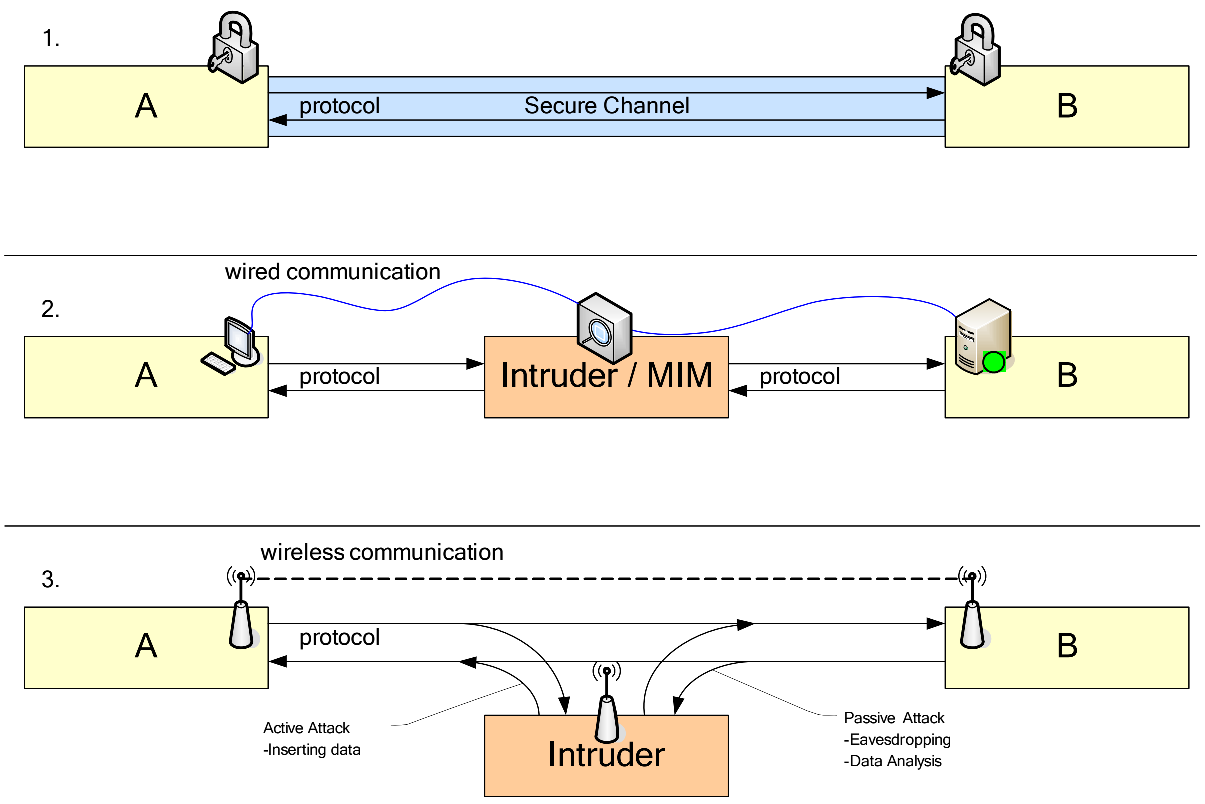


Figure 4: Intruders in different Environments

Figure 4 illustrates three different attacking environments. The first scenario (1) shows a secured channel between *A* and *B* without any intruder. This can either be a physically secured channel or one achieved by encryption. A secured channel is the actually desired state that a security protocol should accomplish. Attacks can have several forms depending on the action the intruder takes. It is considered a passive attack when data is read in transit, whereas active attacks involve messages having to be sent by the intruder. Furthermore, combinations of both forms are also possible.

- ***Passive attacks*** depict the intruder as a probe that is inserted somewhere in the network and has the capability to capture data in transit. The aim of a passive attack is to obtain information from sniffed data, especially that of a confidential nature. Passive attacks are extremely difficult if not impossible to detect since both sender and receiver may never realize that others have had access to the sent messages. Thus, whenever networks are considered as physically insecure or a public network such as the Internet is used, prevention is the only way to confront passive attacks. This is commonly achieved by encrypting messages in order to hide their confidential content. It should be mentioned here that the fact that *A* sends messages to *B* sometimes is sufficient information for an intruder even when the actual data is encrypted. As an example of this one might imagine two companies *A* and *B*; the simple fact that these companies exchange messages, could mean that the amount of data, frequency, time and date of messages could provide more information to a third party *C* than the actual content.

- ***Active attacks*** refer to all attacks where the intruder inserts messages into the network. This can be accomplished for various reasons. Messages do not necessarily have to be generated by the intruder, as he can also use the original captured data. In contrast to passive attacks where prevention is the strategy of defense, active attacks are difficult to perceive. As in wireless environments, active attacks are impossible to prevent and the chosen strategy here is detection. Detection is determined by the circumstances the classification in which a message has or not been sent by the claimed entity. Therefore, this is the realm of authentication and digital signatures [Alk83].

The assumption that can be made on the intruder depends on the environment. Once an intruder compromises an intermediate router, he is able to perform active, passive or any combined means of attack. Figure 4(2) illustrates a scenario where the attacker acts as mediator between *A* and *B*. Here, the intruder can read, store, modify, drop and delay all messages in transit. He is also able to insert messages based on previous information he has extracted as well as redirecting messages to other participants. The variety of possible attacks for such an intruder is endless. One common attack, referred to as *Man in the Middle (MIM)*, describes the situation where the intruder claims to be *A* for *B* and to be *B* for *A*. Figure 4(3) illustrates the wireless scenario which entails slightly different conditions for both the intruder and other principals. Capturing and inserting data is easy for an intruder, whereas delaying, dropping and redirecting messages are more difficult for intruders when a shared wireless medium is used.

An attacker is aware of the fact that he has a vast variety of possibilities to attack protocols. The most common methods are explained hereunder:

Eavesdropping relates to sniffing and the probability of storing messages in transit. This is the most fundamental form of attack and being of a passive nature, needs to be prevented by means of encryption. Eavesdropping is often employed as a basic component of more complex attacks.

Modification involves altering parts of or the entire message in transit. When parts are non-redundant, captured messages can be split, reassembled and inserted in order to break protocols, even when all of individual messages-fields have been encrypted. Data origin authentication and integrity protection for the entire message is pervasive in order to protect authentication and session establishment protocols.

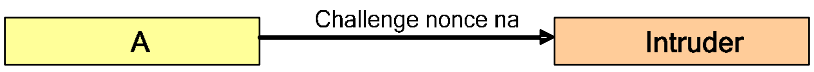
Replay-attacks cover all situations where attackers interfere with a protocol run by inserting messages or parts of messages originally belonging to any previous protocol run. Replay-attacks are also one of the basic protocol breaking tools. These are prevented by timestamps and nonces to guarantee message freshness [Gon93]. Replay protection is essential for session-keys, as such attacks aim to separate keys between the different sessions [Aur97]. A replay-attack counters this command by inserting a previous message in a current protocol run when a timestamp or nonce is not used. For instance, consider a message sent from A to B transferring a session-key K. A has signed the key by encrypting it with its own private key and encrypted it by B's public key to provide secrecy:

$$\left\{\{K\}_{SK(A)}\right\}_{PK(B)}$$

An intruder who captures this message can send this message to B and so B believes he has a session-key with A. This attack says nothing about the capability of the intruder to read messages. However, the intruder can establish the session-key K without him knowing any secret key of A or B.

Reflection- & Interleave-attacks refer to a more complex attack based on a replay-attack. This attack involves two parallel protocol runs or simultaneous sessions. The attacker acts as both the sender and the responder in a challenge response protocol for authentication. An example run is outlined below:

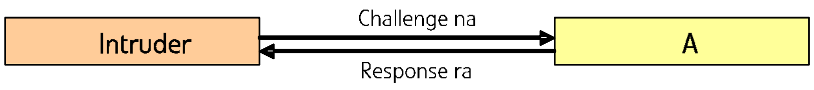
(1) A sends a message to *Intruder I* claiming to be B. A sends a challenge *na* that B should be capable of responding to, since B has knowledge of the secret shared key S.



$$A \rightarrow I : na$$

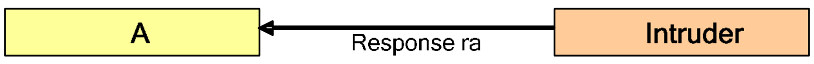
(2) While A waits for the response from I, I sends (reflects) the challenge *na* to A as B would normally do in order to authenticate A.

(3). A computes the response *ra* = *F(na,S)* and sends it back to *I*.



$$I \rightarrow A : na$$
$$A \rightarrow I : ra$$

(4). Upon receiving the response, I again reflects the response to A. A still expects a response from the first protocol run and happily accepts the response from I and thereby authenticates him. Consequently, the protocol is broken.



$$A \rightarrow I : ra$$

In this attack, step 1 and 4 belong to the first (outer) protocol run, while step 2 and 3 belong to the second (inner) protocol run.

In the interleave-attack, *Intruder I* initiates two simultaneous protocol runs in different instances. When the first instance has reached a defined state, the intruder initiates a new second session using information obtained from the first run.

A's viewpoint is that two protocol runs were successfully completed and thus no irregularity occurred which would give him any reason to be suspicious. Interleave-attacks are extremely dangerous as they are very difficult to predict in more complex protocols. The intruder can run protocols multiple times by playing different roles in the attempt to obtain information that would be usable in the main session.

One might not discover interleave-attacks when designing a protocol in which security is only considered by causal deductions. In the example shown above, the causal chain can be formed from the end to the start of the protocol run as follows: (1) *A* finally received a valid response *ra* from somebody. (2) hence, this entity needed to send the response according to the nonce *na* and the secret key *S*. (3) the nonce *na* had recently been sent out and the secret key *K* is only known by trusted parties. The sender must have computed *ra* based on the nonce *na* and the key *K*, thereby indicating that the sender must have knowledge of *K*. (4) Consequently, the sender of *ra* must be a trusted party and is thus authenticated. The authentication is also fresh due to the short lifetime of *na*.

As the interleave-attack illustrates in the example protocol, the authentication actually failed, demonstrating that casual chains are incapable of proofing security. Therefore, this study analyzes protocols that will counter interleave-attacks by means of model checking. Details on formal methods and model checking are given in Chapter 4.

2.5 Protocols for Internet Security

A vast variety of protocols deal with the securing of Internet traffic. The *Internet Protocol Security Suite (IPsec)* [KA98] consists of a number of protocols that perform such tasks as distributing keys, data encryption and authentication.

2.5.1 Oakley and ISAKMP

The *Oakley protocol* [Orm98] uses a hybrid Diffie-Hellman [DH76] technique in order to establish session-keys on Internet hosts and routers. Oakley can either be used independently or when additional attribute negotiation is required, in conjunction with the *Internet Security Association and Key Management Protocol (ISAKMP)* - RFC2408 [Mau98]. ISAKMP is a framework for supporting the negotiation of security attributes, but does not itself provide session-key establishment. However, ISAKMP can utilize a vast range of session-key establishment protocols such as Oakley, in order to provide a solution for complete Internet key management. ISKMP and Oakley, have both been integrated into a hybrid protocol and thus form the basis of the *Internet Key Exchange protocol (IKE)* - RFC 2409 [HC98] introduced in Section 2.5.3. This conjunction offers full security association attribute negotiation and authentication methods that provide both repudiation and non-repudiation. Implementations in the Internet can be used for establishing secure tunnels and connections for securing communications as well as for establishing remote network access. The rules between hosts or security gateways in the Internet are based on a *Security Parameter Index (SPI)*.

Maintained in databases, SPIs cover the following parameters:

- The authentication mechanism can either be based on secret or public key cipher or certificates.
- The encryption mechanism, designates the utilized algorithm, the mode of operation, the key length and the optionally initialisation vector.
- The hash algorithm
- The actual key values and lifetimes.
- The renewal period required for re-establishing the SA.

SPIs are the basis for transferring data securely in the Internet. Thus, they are utilized in order to provide authentication and encryption. SPIs can be established either manually or with the aid of an exchange mechanism. The complete functional facility is covered by the *Internet Security Protocol*.

2.5.2 Internet Security Protocol (IPsec)

The TCP/IP stack was initially used mainly in academic environments, where designers were not particularly concerned about security issues. Nevertheless, the significant growth of the Internet, driven by e-commerce demanded the creation of new services that incorporated sound security provisions. Hence, the *Internet Engineering Task Force (IETF)* established a working group whose objective was to develop an IP security protocol suite. This was named IPsec [KA98]. The function of IPsec is to secure Internet communication at the network layer. IPsec supports both IPv4 and IPv6 [DH98] with a set of extensively configurable security services. IPsec is an obligatory component of IPv6 that succeeded the IPv4. However, the use of IPsec is optional within IPv4.

The tasks of IPsec cover data confidentiality, data integrity, access control, data origin authentication and anti-replay protection.

Due to the fact that IPsec is method-independent, it can be implemented by using a wide range of different cryptographic algorithms.

2.5.2.1 Overview

IPsec is a set of cryptographic protocols for securing packet flows and key exchanges. Two of these packet flow protocols, namely the *Encapsulating Security Payload (ESP)* [KA98a] provides authentication, data confidentiality and message integrity, whereas the *Authentication Header (AH)* [KA98b] provides authentication and message integrity, but does not offer confidentiality. The AH was originally only used for integrity while the ESP was only used for encryption. The function of authentication was subsequently added to the ESP header.

Key establishment and management are crucial tasks for the architecture and are performed by the *Internet Key Exchange (IKE)* [HC98] protocol as introduced in Section 2.5.3.

Standards produced by the IETF are referred to as *Request For Comments (RFCs)*. The connections between IPsec related RFCs are depicted in Figure 5.

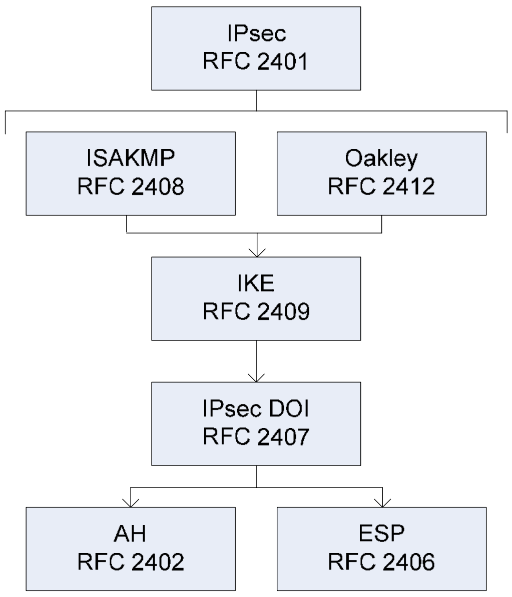


Figure 5: IPsec Related RFCs

2.5.2.2 Modes of Operation

Depending on the type of node, IPsec is intended to operate in two different modes: *Transport Mode* and *Tunnel Mode*. The former simply attaches IPsec extension headers and is commonly used for host to host communication, whereas the latter provides a secure tunnel between two gateways. The tunnel mode can be used to implement portal-to-portal communication security between intermediate nodes. Security is provided for several machines or even to a whole sub-network by a single gateway node that is typically a firewall or router. Traffic destined for a final host, traverses the security gateway. The gateway encapsulates the traffic and forwards it to a second gateway via tunneling. Similar to IP-in-IP encapsulation, the IPsec tunnel involves two IP headers: An inner header containing the original sender and receiver and an outer header where the tunnel endpoints act as sender and receiver. Figure 6 illustrates IP datagrams in transport and tunnel mode. (1) represents a standard non IPsec datagram which is extended by inserting AH in order to authenticate the entire datagram in (2). In (3) a new outer header is attached for transferring the message in tunnel mode. This header authenticates both inner and outer headers; (4) represents the ESP usage in transport mode. The payload and trailer are encrypted and optionally authenticated by the ESP authentication extension; (5) shows ESP in tunnel mode with a new outer header attached. The inner header is encrypted and thus is invisible in transit within the tunnel. (6) Illustrates the case when both AH and ESP are utilized in order to provide authentication for the IP header in transport mode.

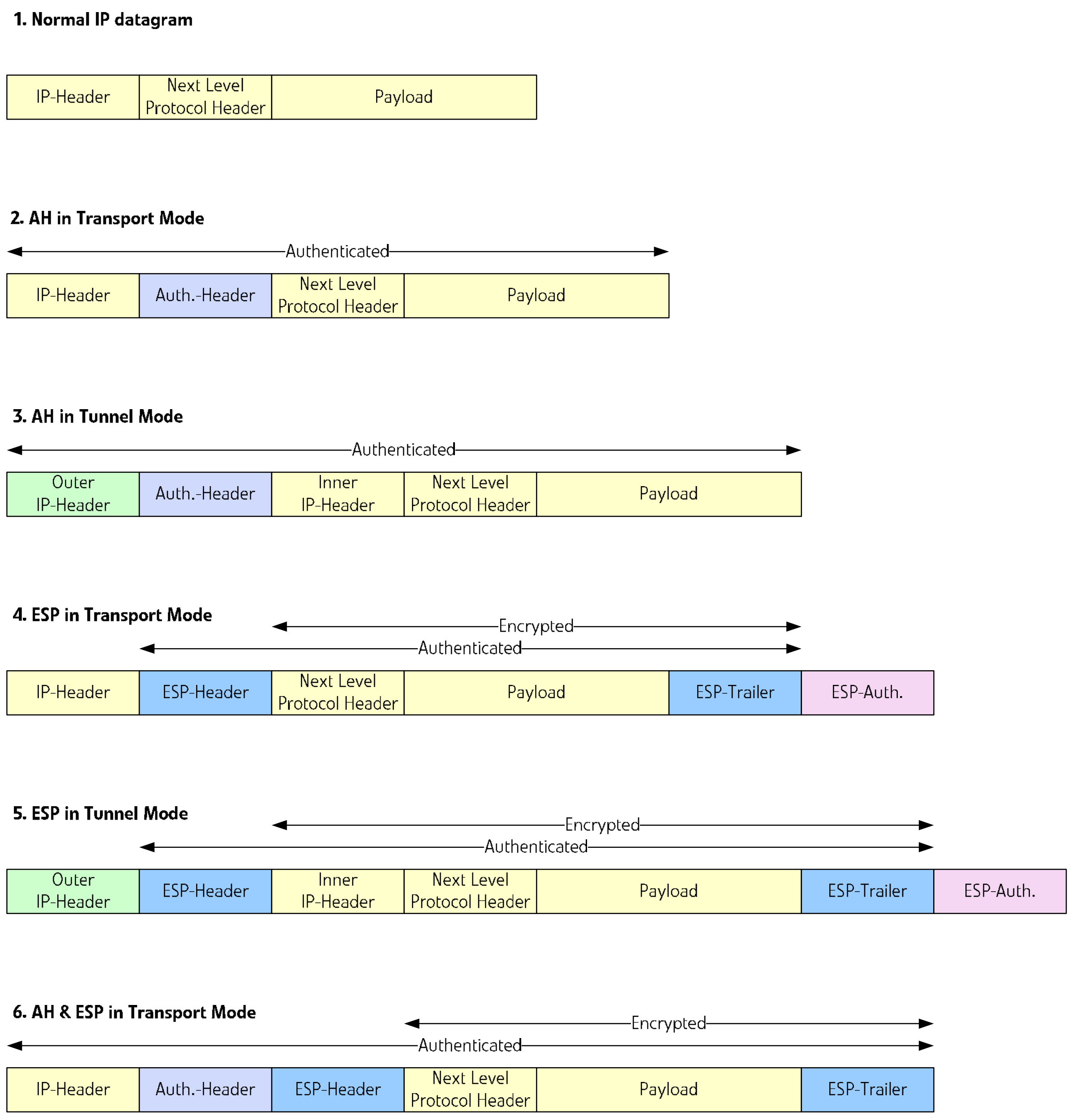


Figure 6: IPsec Modes of Operation

The inner IP header is encrypted and not visible on traverse. A security gateway can be installed on an end point of the connection in order to provide security for network access. When both gateways are on an intermediate node, the tunnel is completely transparent to the endpoints and can thus be used to achieve a *Virtual Private Network (VPN)* [BKW05]. Hence, transport mode in this case is inadequate. Tunnel mode can also be applied for host-to-host communications, but it is especially useful when at least one of the endpoints is a security gateway.

2.5.2.3 Technical Details

IPsec involves two extension headers: The *Authentication Header (AH)* and the *Encapsulated Security Payload (ESP)*. One of them or both can be attached to an IP packet in order to provide security properties.

Authentication Header

The AH is intended to guarantee connectionless integrity and data origin authentication of IP datagrams. It also includes an optional anti replay protection by using the *sliding window* technique and discarding old packets. AH protects all fields of an IP datagram by attaching an AH extension header to the IP packet as illustrated in Figure 7. Fields that change during transfer have been excluded.

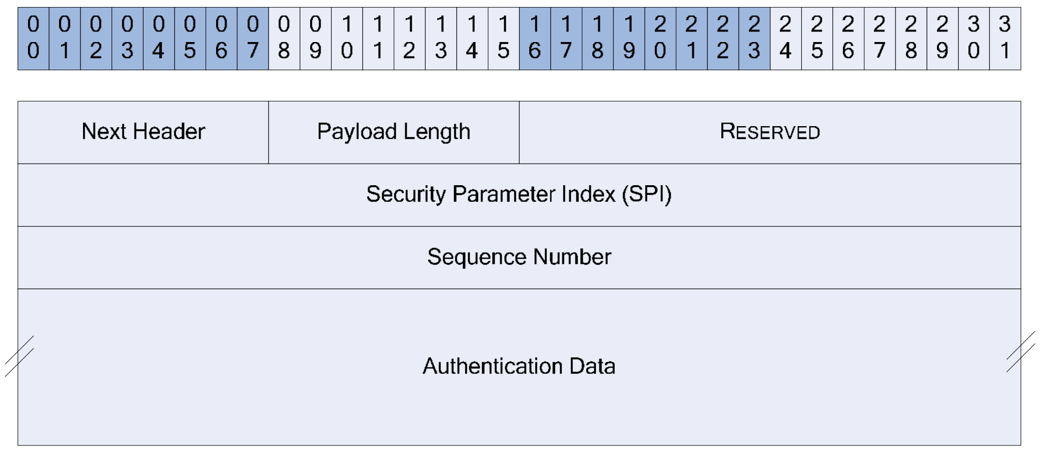


Figure 7: AH Packet Format

The fields of the AH have the following definitions:

- ***Next Header***: Identifies the protocol of the transferred data (e.g. TCP or UDP)
- ***Payload Length***: The overall size of the AH packet.
- ***RESERVED***: Reserved for future use (all zero until used).
- ***Security Parameters Index (SPI)***: Identifies the security parameters in combination with the IP address.
- ***Sequence Number***: A monotonically increasing number for the sliding window in order to prevent replay-attacks.
- ***Authentication Data***: Is an *Integrity Check Value (ICV)* for authenticating the entire IP datagram and is computed by either symmetric or asymmetric encryption algorithms. A hash [MIS95, Rob94] or HMAC [KBC97, MG98, MG98a] as presented in 2.3.6 is commonly used for that purpose.

In order to verify a received datagram, the receiver computes the ICV over all the appropriate fields, filling the mutable IP header field with zeros and using the specified authentication algorithm. When the value attached to the packet coincides with the ICV, then the datagram is verified as having been sent by the claimed sender.

The AH protection in transport mode covers the entire payload, including the upper layer protocol data and all those fields of the IP header which have not been modified in transit. In contrast, the AH in tunnel mode, encapsulates the IP datagram with a new IP header in order to enable the protection to extend to the payload and the whole header.

Encapsulated Security Payload

The ESP extension header, as illustrated in Figure 8, also provides data origin authentication and confidentiality.

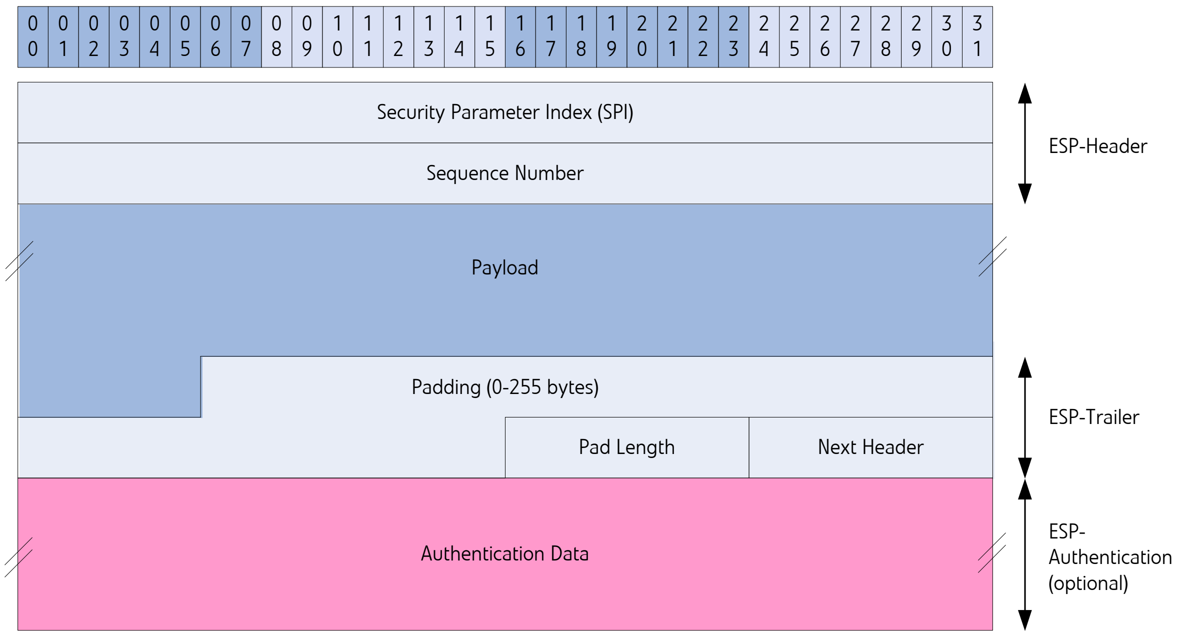


Figure 8: ESP Packet Format

The ESP fields are defined as follows:

- ***Security Parameters Index (SPI)***: Identifies the security parameters in combination with the IP address.
- ***Sequence Number***: A monotonically increasing number, used to prevent replay-attacks.
- ***Payload Data***: The actual data to be transferred which has been either authenticated, encrypted, or both.
- ***Padding***: Used with some block ciphers in order to fit the data into the full length of a block.

- ***Pad Length***: Size of padding in bits.
- ***Next Header***: Identifies the protocol of the transferred data.
- ***Authentication Data***: Contains the data used to authenticate the packet.

In contrast to AH, ESP does not authenticate any IP header field when used in transport mode. However, in tunnel mode, ESP covers the whole IP datagram. Confidentiality is achieved by encrypting the payload plus padding and by inserting the resulting cyphertext in the datagram. Thus, replacing the original plaintext.

Security Databases

The protection of IPsec is based on a set of requirements defined in two databases. The *Security Policy Database (SPD)* and the *Security Association Database (SAD).* The former specifies general policies and is applied to all IP traffic inbound or outbound from a host or security gateway. The latter is more specific and contains parameters associated with an individual connection.

The SPD specifies which security services are on offer to IP datagrams and the method in which they are applied. The database has to be consulted for all inbound and outbound traffic, including that of non-IPsec traffic. The purpose of this is to distinguish between packets that need IPsec processing, those that are allowed to bypass it and packets that should be dropped. This filter acts according to the source and destination address as well as to the corresponding ports.

When an IP datagram is required to be protected by IPsec, a query is made to the SAD. The SAD specifies details on the services that should be provided, as well as the protocols and algorithms to be used. The following entries are required to form a *security association (SA).* This is referred to as a simplex connection between nodes:

- Authentication mechanism for AH
- Encryption mechanism for ESP
- Authentication mechanisms for ESP
- Lifetime of the SA (Time in seconds and maximum amount of traffic in Kbytes)
- Replay protection (only in conjunction with IKE)
- Encapsulation mode (Tunnel or Transport)

Securing a bidirectional data stream between two hosts always involves the establishment of at least two SAs, one in each direction. IPsec SAs are security policies defined for communication between two or more entities; the relationship between the entities is represented by a key.

2.5.3 Key Distribution in IPsec

The Internet Key Exchange (IKE) [HC98] protocol is a mechanism for dynamically establishing SAs between nodes.

IKE as a hybrid protocol, has integrated two earlier security protocols namely OAKLEY [Orm98] and SKEME within an ISAKMP [Mau98] (see Section 2.2.1) TCP/IP-based framework. ISAKMP specifies the framework for key exchange and authentication. The Oakley protocol details a sequence of key exchanges and describes their services (such as identity protection and authentication), whereas the SKEME describes the actual method of key exchange. Although the IKE is not required for IPsec configuration, it offers a number of benefits which include automatic negotiation and authentication, anti-replay protection, *certification authority (CA)* support and the ability to change encryption keys during an IPsec session.

Hence, IKE provides authentication and key agreement during *IKE Phase I* and the IPsec SAs are negotiated in *IKE Phase II* (*Quick Mode*). IKE in *main mode* requires six messages, whereas in *aggressive mode* only three messages are involved.. Diffie-Hellman (see Section 2.4.2.2) is mandatory for providing forward secrecy. The IKE Phase II requires two messages and offers an optional Diffie-Hellman exchange.

A number of authentication modes have been proposed for IKE: authentication with pre-shared secrets, authentication with public key encryption, a revised mode of public key encryption and a signature based authentication. These each have different security properties and computational requirements.

Chapter 3

The Inter-Domain Key Exchange Protocol Approach

This chapter introduces the Inter-Domain Key Exchange Protocol which provides key establishment at the access networks, especially in intra-domain scenarios. It discusses the main assumptions on security associations and trust relationships as well as the requirements for the key establishment at the access network. A detailed view on how the protocol functions is also given.

3.1 Overview

Mobility in wireless networks is based on Access Routers (ARs) providing a service of attaching *Mobile Nodes (MN)* to such ARs. When an MN leaves an AR cell, a handover is required (see Section 2.1.2), which ideally is completely seamless and is not recognizable by the MN due to the MobileIP extensions (see Section 2.2.2). Each AR is connected to a *Local Authentication, Authorization and Accounting (LAAA)* server. Protocols that are widely used request the LAAA to authorize the MN when both ARs are connected to the same LAAA. When the two ARs are working in different domains, the Home AAA (HAAA) server is contacted to authenticate and/or authorize the MN. The time required for performing this HAAA contact may increase the domain handover delay. In order to enhance seamless mobility support, the *Inter-Domain Key Exchange protocol (IDKE)* performs the authentication and session-key establishment locally. The IDKE also aims to reduce the amount of message exchanges on the wireless link and to the Home Network.

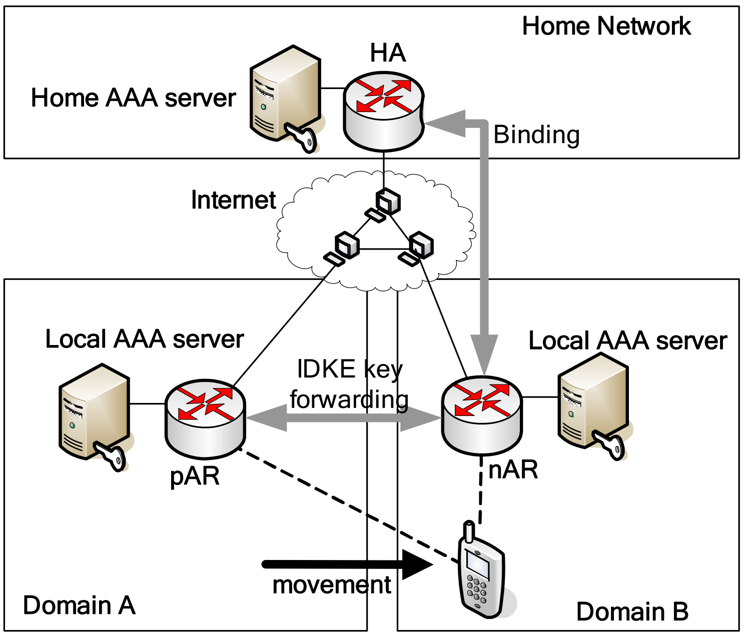


Figure 9: IDKE Key Forwarding Scenario

The basic concept is to transfer all of the required authorization and accounting data from the pAR to the nAR as depicted in Figure 9. In order to achieve this, the nAR needs to be authenticated, authorized and verified trustworthy by the pAR. The pAR and nAR will establish a secure tunnel through which all credentials (e.g. session-key) can be transferred. The pAR also authenticates the nAR to the MN, which enables a session-key to be established between nAR and MN.

IDKE combines well known mechanisms that when separately considered are incapable of providing credential forwarding. Thus, IDKE defines:

- The combination of a token mechanism for initial authentication and trust establishment,
- The IKE (see Section 2.5.3)and IPsec (see Section 2.5.2) for the establishment of a secure tunnel within the ARs and
- The CxTP (see Section 2.2.2.3) for forwarding the credential via the secure tunnel.

3.1.1 Objectives

The purpose of the IDKE protocol is to transfer credentials from pAR to nAR. Credentials refer to confidential data such as cryptographic settings, QoS parameters (see Section 2.1.1) and the session-keys. A huge variety of data can be transferred via the secure tunnel. This document focuses on the symmetric shared session-key which will be referred to as the *Key Session Master Secret (K**SMS)*. *K*SMS This is a mandatory credential which is transferred by the IDKE protocol.

When the IDKE key transfer has successfully taken place, the *K*SMS is then also recognized by the nAR. Hence, the nAR can use *K*SMS for encryption, authentication and access control for the MN. The *K*SMS also acts as a shared secret between pAR, nAR and MN. Therefore, it establishes a trust relationship between all three nodes. Prior to the key transfer, it needs to achieve an initial trust with the ARs. This is carried out by the IDKE authentication mechanism. The mechanism consists of an IDKE authentication token which is sent by the MN and combined with AAA server authentication at the ARs.

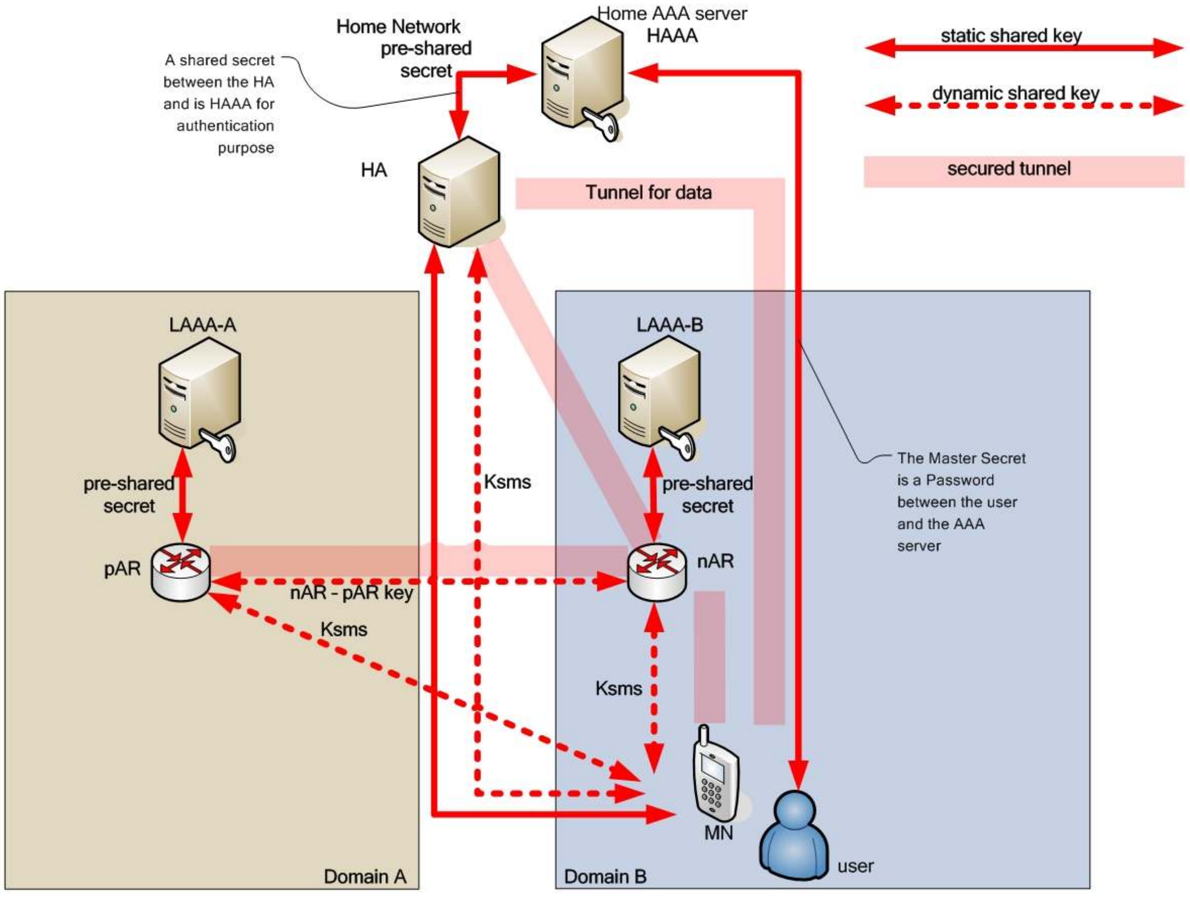


Figure 10: An Overview of Related Trust Relationships and Shared Keys

The nAR's public key is verified by the pAR by requesting its local AAA server. A token is sent by the MN via the nAR to the pAR. This is attached to the initial request message. The token contains an encrypted handover-request to the nAR. If the public key is verified and the token is valid, the nAR is then considered as authenticated.

A major restriction for the MN is the incapability to authenticate the nAR due to low computation power and the missing access to an AAA infrastructure. The ARs also have problems in judging whether an MN actually desires to perform a handover resulting from a claimed request. The IDKE authentication mechanism solves this incapability by combining the MN token for proving actual access to the nAR and the pAR's AAA infrastructure for verifying public keys.

Apart from the establishment of the session-key, the IDKE protocol provides confidence in the knowledge of each other's public keys. This is the basis for generating a fresh session-key (see Section 2.4.1.2) between the MN and the access network based on a Diffie-Hellman like approach (see Section 2.4.2.2). This new fresh session-key K_{NEW} provided by the IDKE protocol is necessary for access control, accounting and electronic payment on the wireless link. K_{NEW} is also used for encryption (see Section 2.3) on the wireless shared link in order to provide confidentiality. The IDKE protocol does not restrict the usage of the key to any protocol or layer. The K_{SMS} can for example, be used as a shared key required by the IPsec extensions. Any data between the MN and the AR is encrypted/authenticated and sent in tunnel or transport mode.

In line with the network topology referred to in Figure 9, the static and dynamic shared keys as well as the secure tunnels are presented in Figure 10. Dynamic keys, commonly referred to as session-keys, are temporarily valid and established only when required. Static keys are assumed to exist prior to the protocol run. The IDKE protocol assumes that symmetric static keys are shared between the ARs and their corresponding AAA servers. The MN (the terminal) and the *User* both share secrets with the Home Network/HA respectively and the HAAA server, as illustrated in Figure 10.

3.1.2 Protocol Framework

The IDKE protocol framework consists of 11 basic and 3 optional messages. Figure 11 illustrates how these 14 messages are separated into five sections. The first message is a router advertisement offering the MN to perform an IDKE based handover. This message is a modified router advertisement that is regularly sent out and thus is not considered to belong to the IDKE protocol.

The first part of the IDKE protocol covers the trust establishment between the ARs based on an authentication token. The second part aims to establish a shared key between the ARs. A Diffie-Hellman like approach as used in IKE is used to provide forward secrecy. Part 3 refers to the key forwarding procedure via a secure tunnel. Part 4 describes the key acknowledgment by means of a handshake between the MN and the nAR. In the optional part 5, the MN performs a binding procedure by sending a binding update to its home network.

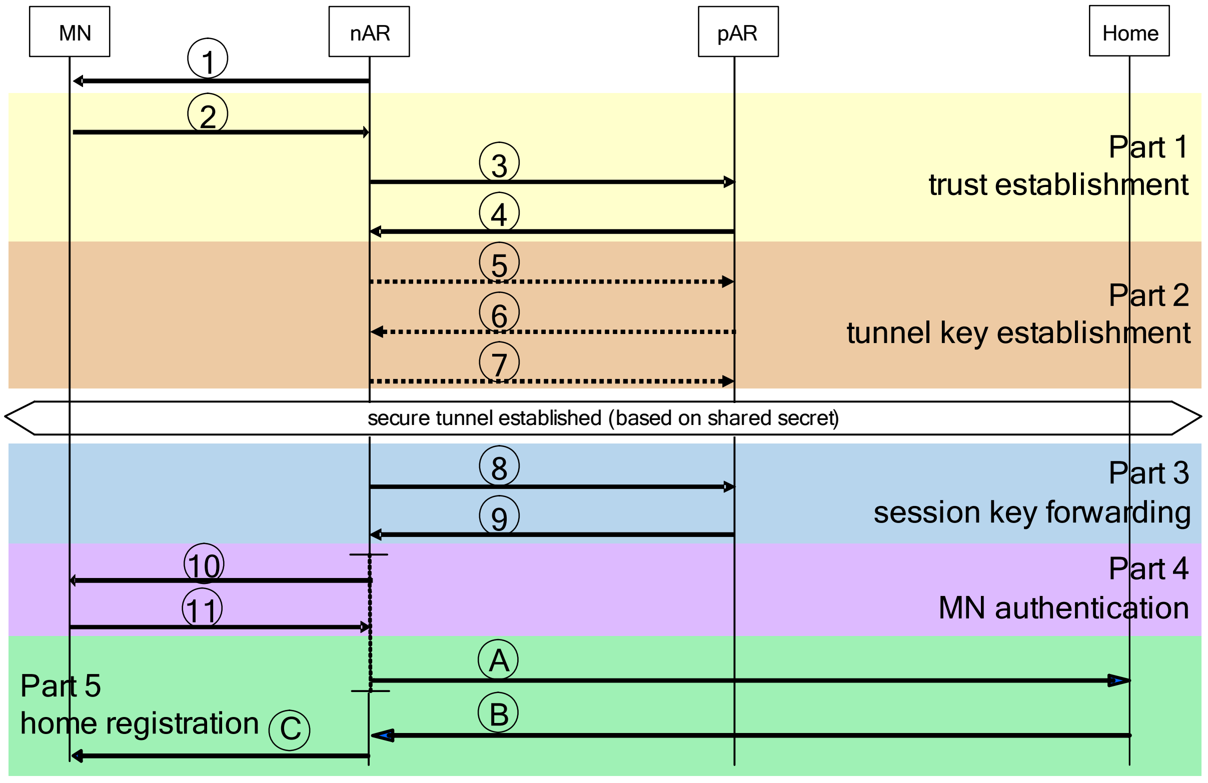


Figure 11: Parts of the IDKE Protocol

New messages introduced by the IDKE protocol are grouped into the following sections:

Router Advertisement: Message 1 refers to the router advertisement offering the MN to perform a handover based on the IDKE protocol.

Part1: The MN creates a token to confirm the nAR's advertisement. This token is sent via the nAR to the pAR in order to authorize the key forwarding.

Part2: A shared key is required to be established between the two ARs. This key should be based on a Diffie-Hellman like approach for providing forward secrecy. The purpose of this key is to provide encryption on a secure tunnel between the ARs. This refers to the combination of IKE and IPsec.

Part3: After successfully establishing the secure tunnel, the session-key K_{SMS} is requested and transferred as suggested by CxTP.

Part4: The nAR has retrieved the K_{SMS}, and thus the nAR is considered trustworthy by the pAR. The nAR then sends an acknowledgement to the MN. Finally, the MN responds to the acknowledgement. Both messages contain parts encrypted by the

K_{SMS} in order to prove to each other their knowledge of the session-key. This handshake may optionally contain a new session-key negotiation procedure.

Part5: The final part is the optional home authentication procedure. This includes the integration of other authentication mechanisms should the key forwarding fail. This option is selected when a higher level of authentication is required.

3.1.3 Preconditions and Postconditions

In order to provide key establishment based on the IDKE protocol, some preconditions need to be fulfilled. The IDKE protocol will otherwise terminate and no key can be exchanged. These preconditions are necessary in order to guarantee the claimed security properties that are referred to as postconditions for the protocol. Thus, the IDKE protocol can be considered as a function

$$IDKE(MN,\ nAR,\ pAR) \rightarrow\ (MN',nAR',pAR')$$

where MN, nAR and pAR fulfill the assumptions (preconditions) and MN', nAR', pAR' will cover all the properties for the postconditions.

3.1.3.1 Preconditions

It is assumed prior to using the IDKE protocol that all entities possess a public/private key pair. However, public keys are not known to all entities. The MN in particular does not have any knowledge of the nAR's public key and vice versa. Thus, initially the MN and the nAR are unable to exchange signed and/or encrypted messages based on public/private keys. They do not share any key at all and the MN is not even aware of the nAR, prior to receiving the router advertisement from the nAR.

The MN and pAR share the fresh session-key K_{SMS} and they know each others public key. K_{SMS} is used for encryption and mutual authentication between the MN and the pAR.

The pAR possesses public keys of all potential nARs or at least should be able to obtain this information by means of an AAA infrastructure. This contains the bundling of identity and corresponding public keys. It is important to mention here that this does not imply that such a potential nAR actually does provide services for the MN. This would only be valid in the case of an intra-domain scenario in which both of the ARs initially trust each other.

3.1.3.2 Postconditions

The MN and the nAR, subsequent to the protocol run, must have established a fresh session-key. This new key is referred to as K_{NEW} and should be known only by the MN and the nAR. K_{NEW} must not have been used prior to the initial message. Furthermore, the MN and the nAR must know each other's public key.

Trust is established between the two ARs and a secured (encrypted and authenticated) tunnel is established. Thus, the pAR and the nAR share and agree on symmetric session-keys for tunneling purposes. The keys are referred to as K_{TUNNEL} and K_{TUNNEL_DH}.

It should be mentioned that the nAR's posterior knowledge has to be the same as the pAR's prior knowledge, since the nAR of protocol run n is the pAR of protocol run $n+1$.

3.2 Protocol Specification

The IDKE framework divides the message flow into five parts: Trust establishment between the ARs, the transfer-key negotiation required for a secure channel establishment, the transfer of K_{SMS} between the ARs and the acknowledgement of the MN as well as the two optional messages for home authentication. In contrast to the IDKE framework in Figure 11, in the protocol specification message 7 and 8 are combined in a single message. Furthermore, the home authentication process is not included. The detailed message exchange is illustrated in Figure 12. The protocol specification of the final version in the standard cryptographic notation is shown in Protocol 1. This classical cryptographic notation is used for all messages as depicted in Section 2.3.2 and 2.3.3 is given in Table 2 that also introduces the Casper protocol notation.

Classical notation	Casper notation	Description
$\{XYZ\}_K$	{*XYZ*}{*K*}	*XYZ* is encrypted by the symmetric shared key *K*
$[X]$	-	*X* is optional
$X/[Y]$	-	*Y* can be optionally used instead of *X*
$PK(X)$	PK(*X*)	asymmetric public key of *X*
$SK(X)$	SK(*X*)	asymmetric secret key of *X*; the inverse key to PK(X)
$\{X\}_{PK(Y)}$	{X}{PK(*Y*)}	*X* encrypted by *Y*'s public key
$\{X\}_{SK(Y)}$	{X}{SK(*Y*)}	*X* signed by *Y*'s secret key
$\{...\}^*$	-	Encryption/signature is optional
$X-ID$	X	unique identifier of *X*
na	Na	nonce *a*, a random number for freshness as introduced in Section 2.4.1.2

Table 2: Cryptographic Notation

The messages of the final IDKE protocol-specification (see Protocol 1) are explained in more detail below:

(1) It is assumed that the MN receives an advertisement, called beacon, from the nAR as well as its public key PK(nAR). This beacon is sent out at regular periods, but it can also be requested by the MN. It also has a domain identifier which enables the MN to judge as to whether a new domain has been offered or not. The MN desires for some reason to re-attach itself to the nAR. There are a number of possible reasons for this. It could for example, be due to the MN's movement or it could have been caused by a connection loss to the pAR.

(2) The MN sends a handover request message to the nAR. The aim of this message is to transfer an authentication token to the nAR. As an authentication token, a part of the message is encrypted by the K_{SMS} and thus cannot be read by the nAR. The purpose of this token is for forwarding it to the nAR in order to authenticate the nAR for the receipt of the session-key K_{SMS}. The authentication token contains the nAR's public key and the identifiers pertaining to the nAR and the MN. Therefore, it only

authenticates the key forwarding to this specific nAR and the purpose is limited to being used only with the initiating MN. The message also includes an encrypted nonce referred to as nonce *na*. The purpose of *na* is to guarantee the freshness and authentication of the key forwarding session. As an alternative to nonce *na* a timestamp can be used. This timestamp remains the same during the entire protocol run and is thus denoted as *TS1*.

1. $\mathrm{nAR} \rightarrow \mathrm{MN} \;: \mathrm{nAR\text{-}ID}, \mathrm{PK}(\mathrm{nAR})$
2. $\mathrm{MN} \rightarrow \mathrm{nAR} \;: \mathrm{pAR\text{-}ID}, \{\mathrm{na/[TS1]}, \mathrm{PK}(\mathrm{nAR}), \mathrm{nAR\text{-}ID}, \mathrm{MN\text{-}ID}\}_{K_{\mathrm{SMS}}}$
3. $\mathrm{nAR} \rightarrow \mathrm{pAR} \;: \mathrm{nAR\text{-}ID}, \{\mathrm{na/[TS1]}, \mathrm{PK}(\mathrm{nAR}), \mathrm{nAR\text{-}ID}, \mathrm{MN\text{-}ID}\}_{K_{\mathrm{SMS}}}$
4. $\mathrm{pAR} \rightarrow \mathrm{nAR} \;: \mathrm{pAR\text{-}ID}, \{\mathrm{na/[TS1]}, K_{\mathrm{TUNNEL}}, \mathrm{pAR\text{-}ID}, \mathrm{nAR\text{-}ID}\}_{\mathrm{PK}(\mathrm{nAR})}$
5. $\mathrm{nAR} \rightarrow \mathrm{pAR} \;: \{\mathrm{na/[TS1]}, \mathrm{nAR\text{-}ID}\}_{K_{\mathrm{TUNNEL}}}$
6. $\mathrm{pAR} \rightarrow \mathrm{nAR} \;: \{\mathrm{na/[TS1]}, \mathrm{H1}\}_{K_{\mathrm{TUNNEL}}}$
7. $\mathrm{nAR} \rightarrow \mathrm{pAR} \;: \{\mathrm{na/[TS1]}, \mathrm{H2}\}_{K_{\mathrm{TUNNEL}}}$

$$\left\langle K_{TUNNEL\_DH} := g^{x^y} \bmod m = g^{y^x} \bmod m \quad | \, H1 := g^x \bmod m, H2 := g^y \bmod m \right\rangle$$

8. $\mathrm{pAR} \rightarrow \mathrm{nAR} \;: \{\mathrm{na/[TS1]}, K_{\mathrm{SMS}}, [\mathrm{PK}(\mathrm{MN})]^*, \mathrm{nAR\text{-}ID}\}_{K_{\mathrm{TUNNEL\_DH}}}$
9. $\mathrm{nAR} \rightarrow \mathrm{MN} \;: \left\{\left\{\{\mathrm{na/[TS1]}, \mathrm{nAR\text{-}ID}, \mathrm{MN\text{-}ID}, [K_{\mathrm{NEW}}]^*\}_{K_{\mathrm{SMS}}}\right\}^*_{\mathrm{SK}(\mathrm{nAR})}\right\}^*_{\mathrm{PK}(\mathrm{MN})}$
10. $\mathrm{MN} \rightarrow \mathrm{nAR} \;: \{\mathrm{na/[TS1]}, \mathrm{MN\text{-}ID}, \mathrm{nAR\text{-}ID}\}_{K_{\mathrm{SMS}}/[K_{\mathrm{NEW}}]}$

*optional

Protocol 1: IDKE Protocol

(3) The nAR forwards the token to the pAR. The pAR checks the token by decrypting the content because pAR initially possesses *KSMS*. The pAR, based on the token content, is able to judge as to whether a key transfer has been authorized or not. The token validity check includes validating all identifiers in order to bind the protocol run to these entities. The pAR verifies whether the MN actually has knowledge of the session-key *KSMS*. Verification of the timestamp respectively the nonce *na* is performed in order to guarantee the freshness of the protocol run. If the pAR considers the nAR is untrustworthy, it will cancel the protocol run. The token combines the nAR's identity, the MN's desire to handover and freshness of this request.

(4) According to the validity of the token, the pAR responds to the nAR. This message authorizes the nAR to start the secure tunnel establishment between the nAR and the pAR. The entire message is encrypted by the verified nAR's public key obtained from the encrypted token. The message includes a key referred to as

K_{TUNNEL}. This is a temporal key used for establishing a secure tunnel between the ARs without providing any forward secrecy.

(5) The nAR sends a message back to the pAR, encrypted by *K_{TUNNEL}*, confirming that it is willing to establish a secure tunnel.

(6) The pAR then starts the tunnel establishment based on *K_{TUNNEL}*. In order to provide forward secrecy for tunnel encryption, the tunnel is established by utilizing an authenticated Diffie-Hellman Key Exchange (DH). Thus, the pAR computes the first so called half-key and sends it to the nAR.

(7) The nAR replies by sending the corresponding second half key encrypted by *K_{TUNNEL}* which also acts as a key request message for *K_{SMS}*. Both, pAR and nAR, are now able to compute a session-key based on DH. This fresh shared key is used for the forwarding tunnel between pAR and nAR is referred to as *K_{TUNNEL_DH}*.

(8) Finally, the pAR forwards the *K_{SMS}*, the nAR's identifier and the MN's public key to the nAR via the secure tunnel.

(9) The nAR sends a message to the MN for the purpose of authentication which contains the initial nonce *na* or timestamp for freshness. The message is encrypted by *K_{SMS}* in order to prove that the key was actually obtained from the pAR. The message is signed by the nAR and encrypted using the nAR's private key. It is then encrypted by the MN's public key in order to make the message only readable to the MN. This is organized as follows:

$$\left\{\left\{\{\text{na/[TS1]}, \text{K}_{\text{NEW}}, \text{nAR-ID}\}_{K_{SMS}}\right\}_{SK(nAR)}\right\}_{PK(MN)}$$

This message also contains a new session-key *K_{NEW}* which provides session-key freshness. Due to the encryption of the session-key, it is only known to the MN and the nAR. *K_{NEW}* is fresh according to *na*; and therefore it must have been created by nAR because of the digital signature.

(10) The MN authenticates itself to the nAR, acknowledges the connection establishment and confirms the reception of *K_{NEW}* by sending back a message. This in turn is encrypted by *K_{NEW}* and contains the nonce *na* or timestamp, as well as the nAR's and MN's identifiers.

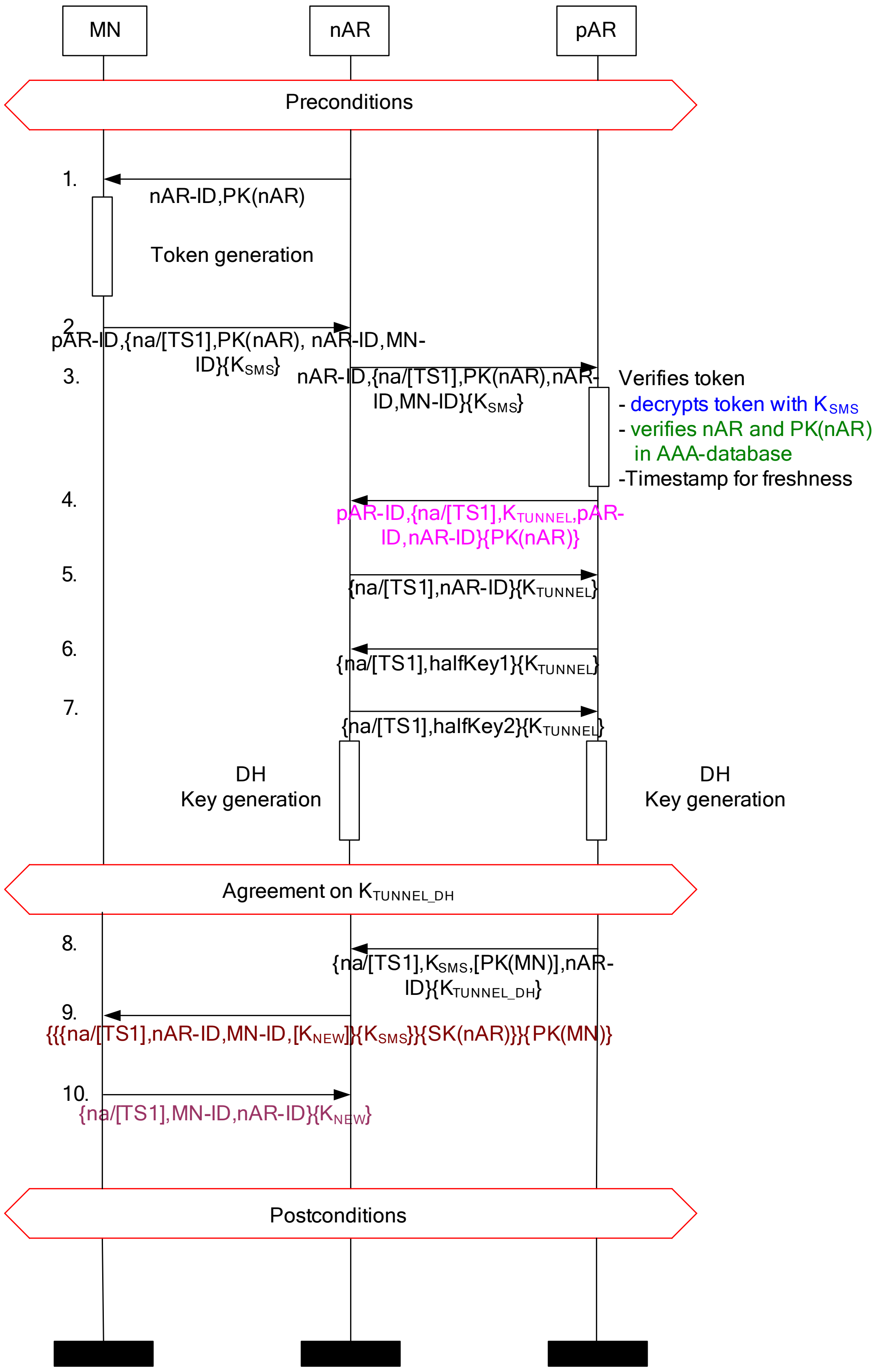

MN
nAR
pAR
Preconditions
1.
nAR-ID,PK(nAR)
Token generation
2.
pAR-ID,{na/[TS1],PK(nAR), nAR-ID,MN-ID}{K_SMS}
3.
nAR-ID,{na/[TS1],PK(nAR),nAR-ID,MN-ID}{K_SMS}
Verifies token
- decrypts token with K_SMS
- verifies nAR and PK(nAR) in AAA-database
-Timestamp for freshness
4.
pAR-ID,{na/[TS1],K_TUNNEL,pAR-ID,nAR-ID}{PK(nAR)}
5.
{na/[TS1],nAR-ID}{K_TUNNEL}
6.
{na/[TS1],halfKey1}{K_TUNNEL}
7.
{na/[TS1],halfKey2}{K_TUNNEL}
DH Key generation
DH Key generation
Agreement on K_TUNNEL_DH
8.
{na/[TS1],K_SMS,[PK(MN)],nAR-ID}{K_TUNNEL_DH}
9.
{{{na/[TS1],nAR-ID,MN-ID,[K_NEW]}{K_SMS}}{SK(nAR)}}{PK(MN)}
10.
{na/[TS1],MN-ID,nAR-ID}{K_NEW}
Postconditions

Figure 12: IDKE Protocol Message Flow

3.2.1 Authentication Properties

A number of authentications take place during the protocol run. The final objective is to achieve mutual authentication between all three entities. None of the entities need to authenticate themselves. There are six cases (two for each entity) which still remain to be considered. An exception is that the pAR is not authenticated for the nAR due to the design of the IDKE protocol. It endeavors to forward keys, but cannot provide any initial key establishment. In order to implement an initial key establishment, this is performed with the aid of the home network which is considered as trustworthy. In comparison with inductive proof, the initial key establishment provides trust establishment at hop 1, whereas IDKE provides trust for hop *n+1* by adopting the assumption that *n* is already trustworthy. This concept is referred to as a chain of trust along the path.

The other five authentications need to be provided during the protocol run. Authentication takes place in the following order: MN to pAR, nAR to pAR, nAR to MN, pAR to nAR and MN to nAR. In Figure 12 messages are highlighted by colors that have the following significance:

🟦	Authentication of MN to pAR by decrypting the token using K_{SMS}.
🟩	Authentication of nAR to pAR by verifying the correctness of the relation between nAR and the claimed public key. This is done at the pAR by requesting the LAAA server. The connection between ARs and AAA servers is assumed to be secure. This is not explicitly considered by the IDKE protocol.
🟥	Authentication of nAR to MN by nAR's use of K_{SMS} encrypting PK(nAR), nAR-ID and *na*.
🟪	Authentication of pAR to nAR by MN responding to nAR's message and therefore, accepting K_{SMS} as a valid key.
🟪	Authentication of MN to nAR by acknowledging message 3 by pAR.

3.2.2 Home Authentication & Initial Key Setup

IDKE is a localized approach for key transfer. However, involving the MN home network in the registration procedure is possible by using an IDKE protocol extension called IDKE_home. This home authentication procedure describes the communication between the MN and the home network. The purpose is to establish the K_{SMS} between the MN and the home network and to authenticate the binding of nAR and MN for the home network. Home authentication is an optional feature of the IDKE. It can be performed when the key forwarding fails or for the initial key establishment. Initial key establishment refers to cases when a pAR does not exist and the session is just being enrolled.

The IDKE_home extension involves the messages (A), (B) and (C) as illustrated in Figure 13. It is important to mention here that the home registration process is not considered in the text on security verification described in Chapter 4. Therefore, the message flow in Figure 12 also does not illustrate any message for the home network and nor does any other message contain data required for the home authentication procedure. The concept is introduced below. The performance evaluations in Chapter 6 analyze the delay caused by this home registration process.

Message (A) is sent from the nAR to the MN's home network and consists of two formulae:

$$MHAT := \left(\left\{nAR\text{-}ID,\ PK(nAR),\ K_{SMS}/[K_{NEW}],TS2\right\}_{MS}\right),$$

$$\left[ART := \left(\left\{\{nAR\text{-}ID, MN\text{-}ID, [...]\}_{SK(nAR)}\right\}_{K_{SMS}/[K_{NEW}]}\right)\right]$$

The first formula relates to *Mobile Home Authentication Token (MHAT).* This is encrypted by a permanent pre-shared key between the MN and the HAAA server and is referred to as *Master Secret (MS)*. The MHAT is computed by the MN according to the router advertisement message. When the MN desires a home authentication, it attaches MHAT to the handover request message. The router advertisement and handover request message are represented by messages (1) and (2) in Figure 13. Hence, the MHAT is forwarded by the nAR, but can never be understood by it. The MHAT includes the nAR's identifier and public key as well as the session-key that is either K_{SMS} or K_{NEW}. The message (A) can also contain a second formula with the title *AR Registration Token (ART)* and which is generated by the nAR. It contains the nAR's and the MN's identifier and may also contain additional data for future purposes such as domain identifier, service identifiers and accounting parameters. The nAR signs the message by its private key and encrypts it by K_{SMS} or optionally *K_{NEW}*. The HAAA server sends back an acknowledgement to the nAR from where it is encrypted by the nAR's public key and the current session-key:

$$\left\{\{\text{AckData},...\}_{\text{PK(nAR)}}\right\}_{\text{K}_{\text{SMS}}/[\text{K}_{\text{NEW}}]}$$

When the home authentication is used for an initial key setup, the nAR does not have any knowledge of the *K_{SMS}* or *K_{NEW}*. Thus, the ART is left blank.

The HAAA server can update the new key *K_{NEW}* for communication between the home network and the MN's access network. When (A) is used for an initial key setup and nAR does not know any session-key, then a new key is setup. The HAAA server generates a random number *ran* and utilizes a function that generates a session based on a random number and *MS*:

$$Initial\_K_{SMS} := funct(\text{MS}, ran)$$

The HAAA server sends back a different message (B) to the nAR which contains the initial session master secret *$Initial\_K_{SMS}$* encrypted by the nAR's public key and *ran* as a challenge (C) for the MN:

$$\{ran,\ Initial\_K_{SMS}\}_{PK(nAR)}$$

As the MN knows *MS,* it is able to compute *$Initial\_K_{SMS}$* on its own. If both nAR and MN finally know *$Initial\_K_{SMS}$,* they will be considered as mutually authenticated. Further authentication can then be performed by the IDKE protocol.

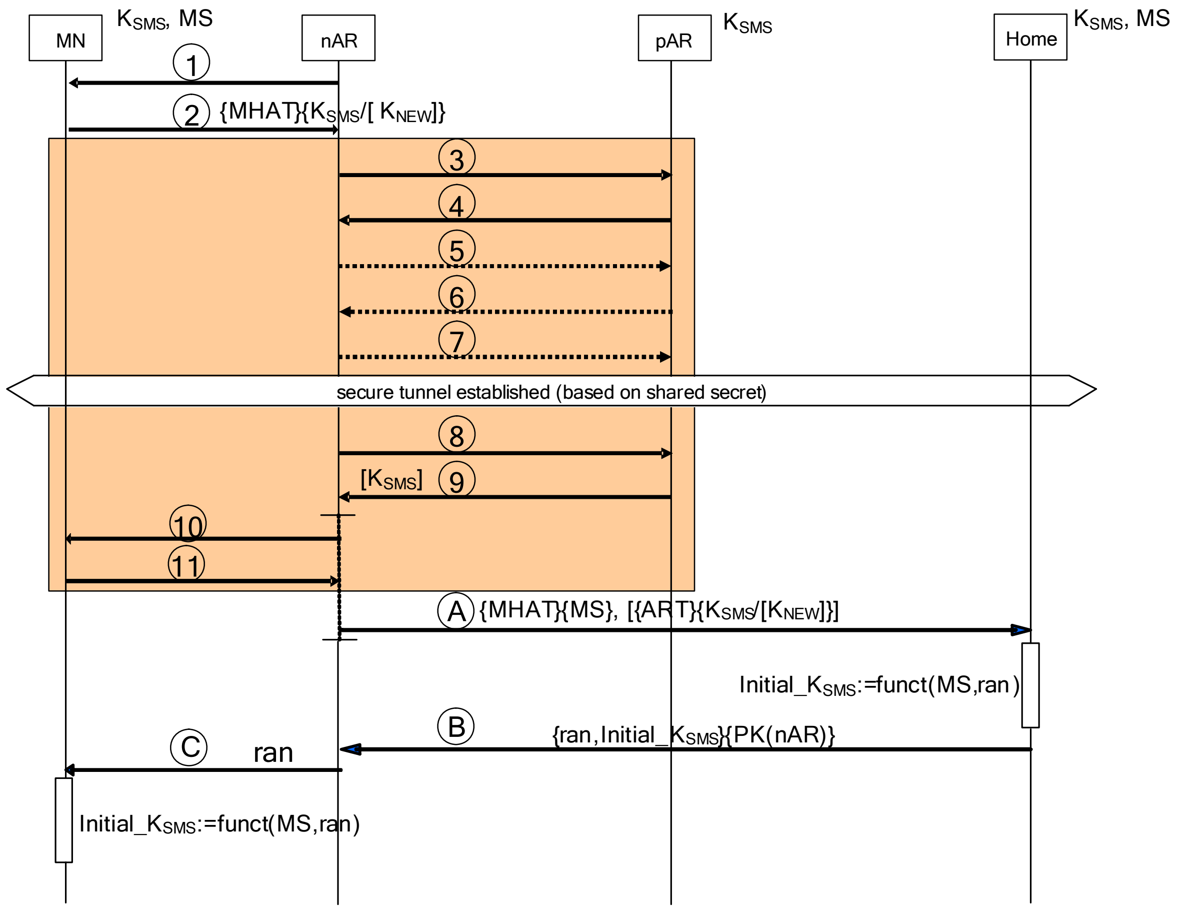

MN
nAR
pAR
Home
KSMS, MS
KSMS
KSMS, MS
1
2 {MHAT}{KSMS/[KNEW]}
3
4
5
6
7
secure tunnel established (based on shared secret)
8
[KSMS] 9
10
11
A {MHAT}{MS}, [{ART}{KSMS/[KNEW]}]
Initial_KSMS:=funct(MS,ran)
B {ran,Initial_KSMS}{PK(nAR)}
C ran
Initial_KSMS:=funct(MS,ran)

Figure 13: Home Authentication Procedure

Chapter 4

Security Verification

This chapter introduces formal security verification methods such as theorem proving and model checking. Firstly, an overview on several approaches is given and their appropriateness for analyzing the IDKE protocol is examined. The chosen approach, a model checker titled Failure Divergence Refinement (FDR) is introduced as well as its underlying algebra, Communicating Sequential Processes (CSP). Secondly, the IDKE protocol is improved by successively analyzing several versions of the IDKE protocol. Finally, the final light weight and secure version of the IDKE protocol is achieved.

4.1 Introduction

4.1.1 Overview

Communication protocols used in distributed systems such as computer networks need to fulfill requirements that guarantee the correct behavior of a protocol. Correctness can be interpreted from a vast variety of viewpoints, for instance considering robustness against unexpected inputs and messages, guaranteeing the absence of any livelock or deadlock, or analyzing interoperability aspects. Whenever verifying the correctness one must be clear on: the objectives of the protocol the provided properties, the considered assumptions, the given environment and the afforded pre- and post-conditions of the protocol. When focusing on security verification, the specification covers authentication or agreements and the secrecy of confidential data. The security protocols can then be verified by formal methods [Gol00] that can be divided into two major categories: model checking and theorem proving. The former explores a huge, but finite space of states in order to find a sequence of transactions in reaching an undesired state. Undesired states refer to all states that do not conform to the protocol objectives. An intruder in such a state knows a secret or he has been authenticated. Once an undesired state has been reached the protocol is broken, whereas the sequence to reach this state is referred to as an attack. Hence, model checkers are able to discover attacks in given environments. The latter approach of theorem proving, is more mathematical since it is based on deduction rules. Theorem proving approaches claim to allow for the checking of a protocol by considering all possible states and thereby formally proving the correctness. However, the assumptions applying to theorem proving methods are sometimes difficult to fulfill. Therefore, protocols that have been stated as being correct have later been discovered to be vulnerable to an attack. Both methods require computer assistance in order to carry out the analyses. Model checking approaches are commonly based on a general purpose model checker and a transformation toolset for handling the dedicated inputs of the security protocols.

This chapter describes the most common approaches and verifies the IDKE protocol by means of model checking. Several versions of the IDKE protocol are actually analyzed for two reasons: Firstly, the reason why the protocol specification was selected. This was discussed in Chapter 3. The final version is selected by presenting attacks on the IDKE protocol modifications that do not fulfill the desired protocol properties. Secondly, apart from the security aspect, the performance decreases due to the increasing number of encrypted or authenticated messages and their size. In order to obtain a lightweight protocol, each message and operation needs to be carefully considered as to whether it is actually required to fulfill the requirements.

A brief overview of related works will initially be given followed by this study's application of the *Communicating Sequential Processes (CSP)* as a basis for the *Failure Divergence Refinement (FDR).* The model checker FDR as well as a CSP-compiler named Casper will be presented in Section 4.2.2 and ultimately, the Casper/FDR tool chain used for security protocol analyses will be introduced.

4.1.2 Tools

Security protocol development increased the demands on verification methods as even small protocols caused problems when being analyzed. Security protocols turned out to be too complex for analyzing without any computer aid [AN95, Boy93, GNY90, Gol99, Gol03]. Thus, a vast variety of formal approaches for security protocol verification such as *Spi-Calculus* [AG99] and *Murϕ* [MMS97] were developed.

Some of these formal methods are described herein. The aim of this section is to determine which approach is the most appropriate one for the IDKE protocol verification. It also shows how the IDKE protocol can be verified by a theorem proving approach termed the *BAN logic*. However, the limitations of the BAN logic as well as the limitations of other approaches has been the incentive for the section that deals with the subject of the FDR model checker.

4.1.2.1 BAN Logic

The logic of *Burrows, Abadi and Needham (BAN)* [BAN89, BAN90, BAN91] was developed in 1989 and is one of the first formal protocol verification approaches. This theorem proving method is used for the reasoning out of authentication and key establishment protocols. Its main advantage is that proofs in BAN logic are simple, short and can be obtained manually.

BAN Expr.	**Interpretation of the expression**
$P \mid\equiv X$	P belives X; P belives that X is true.
$P \triangleleft X$	P sees X; P has received a message from which X can be read.
$P \mid\sim X$	P once said X; P has sent a message containing X.
$P \mid\Rightarrow X$	P has jurisdiction over X; P is trusted on the truth of X.
$\#(X)$	X is fresh; X has not been sent previous to the current protocol run.
$P \xleftrightarrow{K} Q$	P and Q share key K; K is confidential.
$\overset{K}{\mapsto} \mathrm{P}$	P has K as public key; the matching secrect key is K^{-1}
$\{X\}_K$	X is encrypted by K.
$P \overset{X}{\rightleftharpoons} Q$	Formula X is known only to P and Q.
$\langle X \rangle_{\mathrm{Y}}$	X combined with formula Y; Y is assumend to be secret.

Table 3: BAN Logic Expressions

The basic concept is to determine how the belief of agents in other agents evolves whenever new information is received. An idealization process creates formulae containing the initial and the end knowledge of all agents. Furthermore, these formulae contain the assumptions prior to the protocol run and at all stages of a protocol run. The most important factor of the BAN logic is its limitation on *authentication*. This means that there are no secrecy statements which are expressible. It is important to consider the BAN-analysis-results carefully. The interpretation is difficult since BAN explicitly assumes that all participants are honest and are therefore trustworthy. Having these two assumptions in mind, it is not surprising that the BAN logic proved the *Needham Schroeder Public Key Protocol (NSPK)* [NS78] as secure, while *Lowe* outlines an attack [Low96]. Nevertheless, BAN logic is a powerful tool to prove equivalences of protocols or parts of protocols and can be used to optimize over-engineered security protocols to their minimal version.

The syntax of BAN covers three primitive objects that are principals, keys and nonces. Protocol messages are expressed as formulae ranging over *X* and *Y*, where *P* and *Q* stand for principals and *K* range over keys. The formal notation for shared key protocols is illustrated in Table 3.

Proves are based on deduction rules read as "if formulae $X_1,\dots,X_n$ hold than consequently *Y* holds", written more concisely as:

$$\frac{X_1,\dots,X_n}{Y}$$

The BAN logic introduced five interference rules which are:

(1) Message-meaning rule for shared keys:

$$\frac{P \mid\equiv P \xleftrightarrow{K} Q, P \triangleleft \{X\}_K}{P \mid\equiv Q \mid\sim X}$$

P believes that it shares a key *K* with *Q* and sees a message *X* encrypted by *K*. Thus, *P* believes that *Q* had sent message *X*.

(2) Message meaning rule for public/private keys:

$$\frac{P \mid\equiv \overset{K}{\mapsto} Q, P \triangleleft \{X\}_{K^{-1}}}{P \mid\equiv Q \mid\sim X}$$

(3) Freshness rule:

$$\frac{P \mid\equiv \#(X)}{P \mid\equiv \#(X,Y)}$$

If *P* believes that *X* is fresh than *P* believes that *X*, in conjunction with *Y*, is also fresh.

(4) Nonce-verification rule:

$$\frac{P \mid\equiv \#(X), P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

If *P* believes that a nonce *X* is fresh and *P* belies that *Q* had once said *X* than *P* also believes that *Q* believes in nonce *X*. This indicates that *Q* had previously said *X* and was previously alive.

(5) Jurisdiction rule:

$$\frac{P \mid\equiv Q \mid\Rightarrow X, P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$$

In order to analyze the protocol messages, they need to be converted into BAN formulae. Idealizations that need to be performed prior to analyzing the IDKE protocol are illustrated in Table 4. BAN logic instances are represented by a single character, so that the pAR is denoted as *P*, the nAR as *N* and the MN as *M*. The BAN approach then deduces the protocols goals from the following preconditions (called assumptions in BAN logic):

$$\begin{aligned}&\text{IDKE Assumptions:}\\ &M \mid\equiv (M \xleftrightarrow{Ksms} P, \overset{PK(P)}{\mapsto} P, \overset{PK(N)}{\mapsto} N)\\ &P \mid\equiv (M \xleftrightarrow{Ksms} P, \overset{PK(M)}{\mapsto} M)\end{aligned}$$

A simplified IDKE Protocol in standard notation	Idealized BAN formulae
$1.\,\mathrm{nAR} \rightarrow \mathrm{MN}:\mathrm{PK}(\mathrm{nAR}),\mathrm{nAR\text{-}ID}$	---
$2.\,\mathrm{MN} \rightarrow \mathrm{nAR}:\mathrm{pAR\text{-}ID},\{\mathrm{na},\mathrm{PK}(\mathrm{nAR}),\mathrm{nAR\text{-}ID},\mathrm{MN\text{-}ID},\mathrm{pAR\text{-}ID}\}_{\mathrm{K_{SMS}}}$	$\left\{na, \overset{PK(N)}{\mapsto} N\right\}_{K_{SMS}}$
$3.\,\mathrm{nAR} \rightarrow \mathrm{pAR}:\mathrm{nAR\text{-}ID},\{\mathrm{na},\mathrm{PK}(\mathrm{nAR}),\mathrm{nAR\text{-}ID},\mathrm{MN\text{-}ID},\mathrm{pAR\text{-}ID}\}_{\mathrm{K_{SMS}}}$	$\left\{na, \overset{PK(N)}{\mapsto} N\right\}_{K_{SMS}}$
$4.\,\mathrm{pAR} \rightarrow \mathrm{nAR}:\mathrm{pAR\text{-}ID},\{\mathrm{pAR\text{-}ID},\mathrm{nAR\text{-}ID},\mathrm{K_{TUNNEL}}\}_{\mathrm{PK(nAR)}}$	$\{K_{TUNNEL}\}_{PK(N)}$
$5.\,\mathrm{nAR} \rightarrow \mathrm{pAR}:\{\mathrm{nAR\text{-}ID}\}_{\mathrm{K_{TUNNEL}}}$	$\{N\}_{K_{TUNNEL}}$
$6.\,\mathrm{pAR} \rightarrow \mathrm{nAR}:\{\mathrm{H1}\}_{\mathrm{K_{TUNNEL}}}$	$\{H1\}_{K_{TUNNEL}}$
$7.\,\mathrm{nAR} \rightarrow \mathrm{pAR}:\{\mathrm{H2}\}_{\mathrm{K_{TUNNEL}}}$	$\{H2\}_{K_{TUNNEL}}$
$8.\,\mathrm{pAR} \rightarrow \mathrm{nAR}:\{\mathrm{K_{SMS}},\mathrm{PK}(\mathrm{MN}),\mathrm{nAR\text{-}ID}\}_{\mathrm{K_{TUNNEL\_DH}}}$	$\{K_{SMS}, PK(M)\} K_{TUNNEL\_DH}$
$9.\,\mathrm{nAR} \rightarrow \mathrm{MN}:\left\{\left\{\{\mathrm{na},\mathrm{K_{NEW}}\}_{\mathrm{K_{SMS}}}\right\}_{\mathrm{SK(nAR)}}\right\}_{\mathrm{PK(MN)}}$	$\left\{\left\{\{na, K_{NEW}\}_{K_{SMS}}\right\}_{PK(N)^{-1}}\right\}_{PK(M)}$

Table 4: IDKE Message Idealizations of BAN Logic

The IDKE protocol has many goals to achieve. A very simple one will act as an example, illustrating that under the assumption that *M* believes in the public key of the *N*, then the goal that *P* believes in the public key of *N* is fulfilled. This goal is expressed as

$$\begin{aligned}&\text{IDKE Protocol Goal 1:}\\ &P \mid\equiv \overset{PK(N)}{\mapsto} N\end{aligned}$$

The goal is reached by the deduction illustrated by the following proof:

Proof of IDKE Goal 1 :
Starting with message 3 (see protocol)

$$P \triangleleft \{na, \overset{PK(N)}{\mapsto} N\}Ksms$$

using the assumption

$$P \mid\equiv M \xleftrightarrow{Ksms} P$$

applying the message - meaning rule for shared keys (1)

$$\frac{P \mid\equiv M \xleftrightarrow{Ksms} P, P \triangleleft \{na, \overset{PK(N)}{\mapsto} N\}Ksms}{P \mid\equiv M \mid\sim (na, \overset{PK(N)}{\mapsto} N)}$$

based on the jurisdiction rule (5)

$$\frac{P \mid\equiv M \mid\Rightarrow (na, \overset{PK(N)}{\mapsto} N), P \mid\equiv M \mid\equiv (na, \overset{PK(N)}{\mapsto} N)}{P \mid\equiv (na, \overset{PK(N)}{\mapsto} N)}$$

then follows

$$P \mid\equiv (na, \overset{PK(N)}{\mapsto} N) = P \mid\equiv \overset{PK(N)}{\mapsto} N \,\&\, P \mid\equiv na$$

□

It should be mentioned that the IDKE Goal 1 is somewhat obvious and can simply be verified by the BAN logic. However, other proofs are more complex and the details of each of these cover several pages.

A number of protocols have been verified by the BAN logic, such as the Needham-Schroeder public key protocol which is the basis for the Kerberos [Gam03, Tun99] authentication protocol. The main drawback of the BAN logic is that these protocols have later been discovered to be insecure. In conclusion, the BAN logic is inappropriate for use in the formal verification of the IDKE protocol, even if it does present a convenient method. However, the BAN-logic does provide a simple toolset that enables one to analyze protocols instantly in order to obtain an initial impression [BM94, Nes90, HRM+03].

4.1.2.2 NRL Protocol Analyzer

This is a hybrid approach based on both model checking and theorem proving. It was devised by the *US Naval Research Laboratory (NRL)* [Mea96] and is referred to as the *NRL Protocol Analyzer (NPA)*. As a result of a long term project, the NPA is a tool for verifying the security properties of cryptographic protocols [Mea99]. This immense program consisting of several hundred thousands lines of Prolog-code, performs checks based on graphical searches. Starting with insecure states, the tool endeavors to reach the initial state and thereby locate any attack.

All stages of the protocol are represented as conditional rewriting rules that correspond to the data types used. The specialty of this approach is to check secrecy policies. It can check whether the intruder is able to deduce a secret from the rules and its knowledge. The NRL uses a ploy for any authentication or agreements. The participant's belief in the protocol is stored in local variables in order to check as to whether an attacker is able to obtain possession of these variables.

NPA requires user expertise in order to construct the correct word-processing-rules from the protocol specification. Specification errors will obviously result in false assumptions on the secrecy of variables. Automation has been increased to enable the tool to be more user friendly. However, the general public will not profit directly from this research into the NPA due to governmental restrictions on its use.

4.1.2.3 The Inductive Model

The inductive model [Pau98] was introduced in 1998 for use as a formal security protocol verification-method. The concept of this theorem proving approach is based on the use of induction for proving the results of all infinite possible protocol states without having to explicitly examine all of them. The first tool devised by Paulson [Pau98] is named *Isabelle* and is based on a formalism referred to as the *Higher Order Logic (HOL)* [GM93].

Induction is a well known approach that proves the correctness of a formula *F(x)* to be true for each integer x. This is formulated on the basis that *F(0)* is to be established along with a general result of $F(n) \rightarrow F(n+1)$. The inductive proof then concludes that *F(x)* is true for any positive integer x.

All desirable properties are extracted for protocol verification and it is shown that they are preserved under all possible extensions. Thus, this approach is capable of stating whether protocols are insecure. However, it does not explicitly show an attack. This drawback makes the inductive approach impracticable for improving protocols in cases where they are stated as being insecure.

4.2 The Casper/FDR Approach

A model checking approach is selected for analyzing the IDKE protocol which utilizes a tool chain consisting of a compiler for security protocols named Casper, a general purposes model checker called *Failure Divergence Refinement (FDR)* and the graphical front-end CasperFDR. FDR is based on an algebra termed *Communication Sequential Processes (CSP)* which is described in the following section. The FDR and Casper are presented in later sections.

4.2.1 Communicating Sequential Processes

The *Communicating Sequential Processes (CSP)* algebra [RS01, Sch99] developed by Hoare [Hoa85, Hoa96] is a mathematical framework representing systems that consist of entities (processes). These only communicate via the interchange of messages. Therefore, CSP is perfectly appropriate for the simulation of security protocols in that it describes the exchange of messages between the different agents.

4.2.1.1 The CSP Algebra

All communicating entities in CSP are modeled as *processes* interacting via elements of a set of all visible *events* or *actions* Σ. In contrast to visible events, the internal action is conventionally written τ. The differentiation between visible and invisible events is very important. It is a requirement of the underlying mathematical model of the CSP that equivalence is defined as: "The fact that two different programs produce patterns of actions that cannot be distinguished by an observer."

The minimal program merely contains the *Stop* process that simply does nothing. This process is required in order to symbolize the total end of a chain of actions. By using the *prefixing operator* $"\rightarrow"$ actions can be assigned to processes, e.g., $a \rightarrow P$ reads as "the process performs a and then behaves like P". Based on these fundamentals, a first chain of actions (event *in,* event *out*, event *Stop*) can be constructed:

$$hy, fer \in \Sigma$$
$$hy \rightarrow fer \rightarrow Stop$$

By using these rules, more complex structures such as recursive assignments can easily be built:

Simple: $P = hy \rightarrow fer \rightarrow P$ Pair of processes: $\begin{array}{l} A = hy \rightarrow B \\ B = fer \rightarrow hy \rightarrow A \end{array}$

Instead of exactly specifying the possible action, a set $A \subseteq \Sigma$ of visible actions may be given. The expression $?x : A \rightarrow P(x)$ offers the environment the means of selecting one action x from the set A. After processing x the program behaves like $P(x)$. It is very important to recognize that the environment selects the action from the set, whereas the process itself does not have any direct influence. Hence, x can be thought of as a parameter of P. The following example machine has an internal state s and keys $L \cup \{off\}$ where L is the alphabet of individual characters. *Machine1* illustrates how parameters can be introduced by direct input:

$$Machine1(s) = ?x : L \cup \{off\} \rightarrow Machine1'(s,x)$$
$$Machine1'(s,off) = Stop$$
$$Machine1'(s,x) = crypt(s,x) \rightarrow Machine1(nextstate(s,x))$$
$$(x \in L)$$

Machine1 processes action *x*, which leads to one of the two different states. If *x* is *off*, the machine is turned off and thus process *stop* is reached, otherwise *Machine1* goes to a next state *s* resulting in a function of $nextstate(s,x)$. Presumably, $"nextstate(s,x)" \notin L$ is employed in order to avoid confusing the input and the output of the machine. CSP allows for the adding of channels *in* and *out* to the machine to enable it to handle in- and outputs. The "?" models input while "!" represents the output. The machine presented above without the stop-function will look as follows:

$$Machine2(s) = in?x \rightarrow out!crypt(s,x) \rightarrow Machine2(nextstate(s,x))$$

Machine2 is more natural than *Machine1*, but it lacks the stop function. Thus, an additional important operator is the *choice operator* "□". By adding the operation, the Machine would then appear as follows:

$$\begin{aligned} Machine3(s) = in?x \rightarrow out!crypt(s,x) \rightarrow Machine3(nextstate(s,x)) \\ \square\, off \rightarrow Stop \end{aligned}$$

While all previous assignments were *deterministic*, the choice operator enables *nondeterministic* behavior to be added to the system. However, the developer of the system is unable to predict the actual behavior. The system is allowed to make internal decisions that affect the external view of it - the visible events. For instance, nondeterminism is given by

$$(a \rightarrow a \rightarrow Stop) \square (a \rightarrow b \rightarrow Stop)$$

After the action *a,* the implantations are able to either select *a* or *b* as the following action. This behavior is unpredictable since the sequence of visible events can either be *a,a* or *a,b*. In order to express nondeterminism explicitly, the *nondeterministic choice operator "⊓"* is used. *P⊓Q* and $P \square Q$ act like *P* or like *Q* and have the same set of *traces* that are in all sequences of visible communication. Hence, they can perform $(P \square Q) = trace(P) \cup trace(Q)$. The difference between both is that in the case of *P* □ *Q,* the implementer of a system does not have any influence on which option another user will take. However, in the case of the nondeterministic choice *P⊓Q,* even the user does not have any control over which behavior is actually selected as this function is carried out by the system itself. Another important difference between □ and ⊓ is that relating to the deadlock behavior:

$(a \rightarrow P) \sqcap Stop$ is not the same as $(a \rightarrow P) \square Stop$

The two processes differ since the first one can choose to stop immediately (deadlock), whereas the second behaves like $(a \to P)$. This is because it offers the environment the choice between $(a \to P)$'s actions. However, it does not offer any other possibilities e.g. (*Stop*). The result is in fact the same as merely offering $(a \to P)$'s.

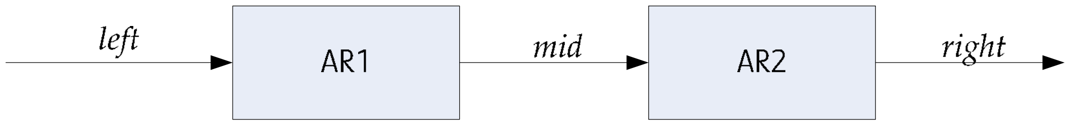
It can be shown that all CSP operators comply with the distributive law, resulting in the equivalence of

$a \to (b \to Stop \Box c \to Stop)$ and $(a \to b \to Stop) \Box (a \to c \to Stop)$.

It is often useful to have generalized versions of the choice operators, e.g. $\Box S$ offers the choice of all processes in *S*, whereas $\sqcap S$ can choose to act like any member of *S*. Therefore, it is necessary that *S* is unequal to the empty set, $S \neq \varnothing$.

An essential concept embracing nondeterminism is *refinement*. Refinement refers to cases where a process *P* has at least the same set of choices as *Q*. This implies the equivalence of *P* and $P \sqcap Q$. It is said "*Q refines P*" written as $P \sqsubseteq Q$. The main strength of the model checker FDR (see Section 4.2.2) lies in the decision on refinement.

The state of the entire system is composed of one state for each component. This fact exponentially increases the state space within the size of the network. However the components' states are not independent of each other. The state of a sender for example is somehow related to the state of the receiver. In order to provide a handshaken communication, it is necessary to synchronize the processes. Therefore *parallel operators* provide synchronization for multiple processes on dedicated events. This concept is important in providing a handshaken communication between processes. The operator $P \parallel Q$ forces all visible actions of *P* and *Q* to be synchronized. Thus, $P \parallel Q$ is only able to perform $a \in \Sigma$, if both *P* and *Q* can. It is also only possible to restrict the synchronization on a subset $X \subset \Sigma$ by using the *interfaced parallel operator* $P \underset{X}{\parallel} Q$. One can perform unsynchronized actions for all other events that do not include *X*, *Q* and *P*. The following example illustrates the parallel operator used for synchronizing a sending AR with a receiving AR:



$$AR1 = left?x \to mid!x \to AR1$$
$$AR2 = mid?x \to right!x \to AR2$$

This formula shows the input choice prefix "?" which denotes the reception of a message, whereas the output choice prefix "!" denotes the sending of a message. The synchronizing *interface* between these processes *AR1* and *AR2* is the channel *mid*.

Thus, it can be denoted by the interfaced parallel operator as $AR3 = AR1 \underset{\{mid\}}{||} AR2$. *AR1* accepts a value *x* of the channel *left*, which in turn is transferred via the channel *mid* to *AR2*. This is then outputted to channel *right* by $right!x$. The combination of multiple processes using the *interfaced parallel operator* increases the complexity of the interface definition and is a potential source of implementation errors. The CSP for this reason provides the concept of *alphabets*. An alphabet is a set of events which are controlled by the process. The interface between two processes is merely the intersection of their alphabets. An action of a chain of interfaced processes can only be processed when all processes agree to whom owns the alphabet set of events. The binary interface operator is written as $P_X \, ||_Y \, Q$ with $X = alphabet\,of\,P$, $Y = alphabet\,of\,Q$. The general form of this operator is

$$||_{i=1}^{n} \left(P_i, A_i\right)$$

defined via induction for $n \geq 2$ by:

$$||_{i=1}^{n+1} \left(P_i, A_i\right) = \left(||_{i=1}^{n} \left(P_i, A_i\right)\right)_{A_n^*} \, ||_{A_{n+1}} \, \left(P_{n+1}, A_{n+1}\right), \quad A_n^* = \bigcup_{i=1}^{n} A_i$$

If the intersection of alphabets is $\varnothing$, concurrency can be expressed by the usage of the *interleaving operator* $|||$ which is an abbreviation for $\underset{\varnothing}{||}$. The general version of the interleaving operator is $|||S$ and this can be used to simplify the interfacing of disjointed sets.

The expansion of networks with an infinite state space can be created by using the parallel operator:

$$P = a \to \left(P \, || \, P\right)$$

There is one additional process *P* in the system for each event *a* performed.

It is sometimes necessary to hide events from the user. In order to achieve this in CSP events of Σ, they are changed into the invisible element named τ. This operation is performed by the *hiding operator* $P \setminus X$. The process $P \setminus X$ behaves in the same way as *P* with the exception that all events from *X* are substituted by the invisible element τ. The updated example of the ARs is then expressed as follows:

$$AR1 = left?x \to mid!x \to AR1$$
$$AR2 = mid?x \to right!x \to AR2$$
$$AR3 = AR1 \underset{\{mid\}}{||} AR2$$

$AR3 \setminus \{mid\}$ only leaves the external communication visible and hides the transfers of values *x* from *AR1* to *AR2*.

When a system is viewed externally, there are a vast variety of reasons why a process ceases to communicate with the environment. It can be caused by reaching the *Stop* state, deadlocking or by being in a certain race condition. It is important to know whether a process has terminated successfully or not, as well as being able to recognize whether a process has reached the *Stop* state. A successful termination is thus denoted by the symbol" $\surd$ ". This represents a special event that has been introduced via the process *Skip*.

$$a \rightarrow b \rightarrow Skip$$

Symbol	**Description**
$Stop$	the process that does nothing
$a \rightarrow P$	event prefix
$?x : A \rightarrow P$	event prefix choice
$c?x : A \rightarrow P$	input prefix choice
$c!x : A \rightarrow P$	output prefix choice
$P \Box Q$	choice between two processes
$\Box S$	general choice
$P \sqcap Q$	nondeterministic choice
$\sqcap S$	general nondeterministic choice
$P \parallel Q$	lockstep parallel
$P_X \parallel_Y Q$	synchronized parallel
$P \underset{X}{\parallel} Q$	interface parallel
$\vert\vert\vert\, S$	general interleaving
$P \setminus X$	event hiding
$P[\![R]\!]$	process relational renaming
$Skip$	successful termination
$P;Q$	sequential composition
$P = F(P)$	recursive definition
$\mu p.F(p)$	recursive process

Table 5: CSP Operator Overview

The example above is a process that successfully terminates after the events a and b have been performed. The use of the parallel operator $P \parallel Skip$ results in a behavior just like P, because the right-hand process terminates immediately and waits for P to do so too. The basic operators of CSP are summarized in Table 5. This table has mostly been imported from [RS01].

4.2.1.2 Process Behavior

In order to be able to understand the CSP process behavior, it is important to have knowledge of the concept of traces. As mentioned earlier, the traces of a process *P* are sequences of visible communications which they can perform. Traces of *Stop* for example are given by:

$$traces(Stop) = \{\langle\ \rangle\}$$

where $\langle\ \rangle$ is the empty sequence. However, the most interesting concern those traces of a process which perform actions where *s^t* is the concatenation of *s* and *t* while s^n is the concatenation of *n* copies of *s*.

$$traces(\mu P.a \to P \Box b \to Skip) = \left\{\langle a\rangle^n, \langle a\rangle^n \wedge \langle b\rangle, \langle a\rangle^n \wedge \langle b, \surd\rangle \mid n \in \mathbb{N}\right\}$$

CSP represents each process *P* by its own set of traces that are:

- nonempty, as the empty trace can be performed by any process
- prefix-closed, that if *s^t* is a trace then so is *s*.

All traces span over a finite set of sequences of Σ that may also include $\surd$.

All words (constructs) of the CSP language are deduced from a basic set of rules used for calculating $traces(P)$ for all *P*=CSP-terms. The following rules illustrate the trace deductions. A complete overview is shown: [RS01] pp. 63, 64.

- $traces(Stop) = \{\langle\ \rangle\}$, the empty trace.
- $traces(a \to P) = \{\langle\ \rangle\} \cup \{\langle a\rangle \wedge s \mid s \in traces(P)\}$, there are two possible behaviors for this process: Either it does nothing or its first event is *a* followed by a trace of *P*.
- $traces(P \Box Q) = traces(P) \cup traces(Q)$, this process offers the traces of *P* and *Q* as is described by the operator $\Box$.
- $traces(\Box S) = \bigcup\{traces(P) \mid P \in S\}$, all traces for any nonempty set *P* of *S*.
- $traces(P \,||\, Q) = traces(P) \cap traces(Q)$, because of the need for synchronization of each event, every trace of the combination has to be a trace of both *P* and *Q*.
- $traces\left(P \underset{X}{||} Q\right) = \bigcup\left\{s \underset{X}{||} t \mid s \in traces(P) \wedge t \in traces(Q)\right\}$, where $s \underset{X}{||} t$ is the set of traces that can result from *P* and *Q* respectively performing *s* and *t*. Details on the calculation are shown on [Ros97] or [Sch99].
- $traces(P \setminus X) = \{s \setminus X \mid s \in traces(P)\}$, where $s \setminus X = s \,\big|\, \left(\Sigma^{\surd} \setminus X\right)$.

When all traces of P satisfy a logical property $S(tr)$ then these traces can be written as $P\,\mathbf{sat}\,S(tr)$ which is equivalent to $\forall tr \in traces(P) \bullet S(tr)$. Three direct consequences result from this definition:

- $P\,\mathbf{sat}\,true(tr)$ for any process *P*. This expresses that every process meets the weakest specification that does not disallow any behavior.
- $(P\,\mathbf{sat}\,S(tr) \wedge P\,\mathbf{sat}\,T(tr)) \Rightarrow (P\,\mathbf{sat}\,S(tr) \wedge T(tr))$, which allows the separate establishment of conjuncts in a specification with a number of conjunctions.
- $(P\,\mathbf{sat}\,S(tr) \wedge (S(tr) \Rightarrow T(tr))) \Rightarrow P\,\mathbf{sat}\,T(tr)$. If *P* satisfies a specification *S*, then *P* satisfies any weaker specification *T*.

4.2.1.3 Discrete Time

Many protocols require a timing-construct in order to provide timestamps that are necessary in avoiding deadlocks or preventing replay-attacks. There are two approaches for including time into the CSP process algebra. The first approach is *Timed CSP* [Sch99] which attaches a nonnegative real number to each event in the set of traces. Therefore, it is possible to record the moment at which an event occurs. The problem with this approach is that it requires a lot of unpredictable changes in the theory of concurrent systems. The second approach is due to this, used in the construction of network protocols by employing CSP (detailed information in [Ros97]): By adopting this method, the alphabet Σ is extended by an extra event *time* in which it is assumed that it occurs at regular intervals. This event is usually termed *tock*. Timing is now possible in the form that any event which takes place between the third and the fourth *tock* is referred to as occurring between time unit *three* and *four*. These time units can be adapted to the system. The following example illustrates a possible usage of the *tock* event. While *a* occurs every time unit in T_1 ,T_2 allows a nondeterministic choice to wait.

$T_1 = a \rightarrow tock \rightarrow T_1$
$T_2 = (a \rightarrow tock \rightarrow T_2) \square (tock \rightarrow T_2)$

The process states can generally be categorized into three groups:

- An *idle* state allows the choice to wait without changing its internal state.
- An *evolving* state allows the choice to wait, but changes its internal state, e.g., a counter.

An *urgent* state is one in which no *tock* is possible. The process is waiting for a further event to occur.

4.2.1.4 Modeling a Security Protocol in CSP

The process behavior for security process modeling needs to be defined and specified in the CSP. Due to the fact that all communication is realized by visible events, the protocol can be modeled as follows: The first step is to obtain a detailed description of the protocol and to extrapolate any agent's view of the message exchange. All inputs and outputs take the form of *receive.a.b.m* and *send.a.b.m*. In each case, *a* and *b* are the participating agents and *m* the message. One should not assume that the agents always handshake on such events in the same manner as CSP's parallel operator || does. In fact it is better to imagine that the messages posted into the environment are similar to throwing a message in a bottle into the sea. Thus, the simplified protocol description (*M* for MN-ID, *N* for nAR-ID and *P* for pAR-ID) for the first three messages of the IDKE protocol looks like this:

N's view (as initiator)

Message 1: N sends to M: $N, PK(N)$

Message 2: N gets from 'M': $P, \left(\{PK(N), N, M\}_{K_{SMS}}\right) \rightarrow MN\_Token$

Message 3: N sends to P: N, MN_Token

.

M's view (as receiver)

Message 1: M gets from 'N': $N, PK(N)$

Message 2: M sends to N : $P, \left(\{PK(N), N, M\}_{K_{SMS}}\right)$

.

P's view (as server)

Message 3: P gets from 'N': N, MN_Token

The quotes '*X*' are to emphasize that the agent can neither be certain of the identity of its partner nor as to whether the messages in fact successfully arrive.

The next step in modeling the protocol, is the transformation of the description into the CSP processes that support the two basic functions *send* and *receive*. However, before transmitting messages, agents have to calculate all of the necessary information, e.g., nonces, hash- or encryption functions etc. In practice (in the real world), nonces are chosen by pseudo-random-number-generators. Due to the large key size respectively long nonces, both are assumed to be unique. However, CSP systems only provide short lengths for keys and nonces and consequently uniqueness is actually not provided. Therefore, another approach must be applied in order to guarantee this. There are two possible solutions to this problem:

- Every process is equipped with a list of all variables.
- An additional process manages key/nonce-creation and thereby ensures uniqueness.

The latter solution is well-suited for the CSP process algebra.

The application of encryption and decryption in the protocol requires processes for encrypting or decrypting messages. There are two different means for an agent to be able to receive encrypted information: One is if the key is in the agent's knowledge, then decryption is possible. The other is when the agent does not have any possibility in deciding whether this message belongs to the protocol or it is in fact just garbage. Therefore, it is not necessary to implement any mathematical encryption/decryption functions into the CSP, but merely to accept or discard received messages by means of an external choice for all acceptable messages. This is based on an agent's knowledge.

$$\Box_{K_{NEW} \in Key} receive.P.N.\left(\{P.K_{NEW}\}_{K_{SMS}} \cdot \{na\}_{K_{NEW}}\right) \rightarrow P(N,P,nb,K_{NEW})$$

The statement above illustrates a message acceptance of the form $P \rightarrow N : \{P.K_{NEW}\}_{K_{SMS}} \cdot \{na\}_{K_{NEW}}$. The addressee *N* is able to decrypt the message and the key K_{NEW} is included as an input into the following process. *N* has decrypted the message by K_{NEW} (which must have had prior knowledge) and has consequently gathered the key K_{NEW}. In order to avoid deadlocks, the above statement can alternatively be implemented as follows:

$$\begin{array}{l} \Box_{K_{NEW} \in Key} receive.P.N.\left(\{P.K_{NEW}\}_{K_{SMS}} \cdot \{na\}_{K_{NEW}}\right) \rightarrow P(N,P,nb,K_{NEW}) \\ \Box AbortRun(N) \end{array}$$

If acceptance is impossible, the *AbortRun* action will be performed, thus avoiding a deadlock. By using these rules, a complete set of processes can be implemented representing *Initiator*, *Responder* and *Server*. A possible process implementation can for instance be:

$$\begin{array}{l} Initiator(N,na) = \\ env?P : Agent \rightarrow send.N.P.N.na \\ \quad \underset{\substack{K_{SMS} \in Key \\ nb \in Nonce \\ m \in T}}{\Box} \left(\begin{array}{c} receive.M.N.\{nb\}_{K_{SMS}}.m \rightarrow \\ send.N.P.m.\{nb\}_{K_{SMS}} \rightarrow Session(N,P,K_{SMS},na,nb) \end{array} \right) \end{array}$$

T is the set of all objects that *N* can accept as messages. The initial communication $env?P:Agent$ is a representation of how the process's local environment might inform it to open up a session with agent *P*. Nothing prevents $N \in Agents$ from communicating with itself. This is not superfluous, because of the possibility of an identity having a number of active processes. If this is undesired, the CSP programmer is free to modify $env?P:Agent$ to $env?P:Agent\setminus\{N\}$ which explicitly excludes the agent.

The most usual behavior an agent can perform is the ability to start the protocol in multiple instances by running it in both the role of sender- and receiver. This can be specified using the generalized interleave operator $|||_{i\in I}\ P_i$, with each single protocol run as one of the P_i.

4.2.1.5 Modeling an Intruder

There are two possibilities for implementing the intruder: *Lazy spy* or *perfect spy*. The former refers to an attacker that is absolutely passive and by means of capturing data he endeavors to obtain certain secret information. However, the perfect spy does everything in combining passive and active attacks in order to discover any vulnerability (see Section 2.4.3). Thus, when aiming to verify protocol security, the more appropriate approach is to model an intruder as a perfect spy.

One approach is to implement the intruder as an additional CSP process. However, an implementation could be difficult and although the set of attacks might be reduced, it would only be possible to detect attacks following routes that are anticipated by the intruder's definition. Hence, the real solution is both elegant and simple: The intruder is modeled as the environment, which was initially assumed to behave in a nondeterministic manner and not being goal-oriented. A number of "intruder instances" are assumed to be joined to one single intruder who is able to perform an action that is cryptographically justifiable:

- He can overhear and/or block messages from any agent.
- He can generate any message that can be built on the basis of the intruder's knowledge including garbage messages (without any information).
- He can act as any agent other than those that are explicitly built into the network.

At the CSP level this implementation leads to:

$$\begin{aligned} Intruder(X) &= learn?m:messages \rightarrow Intruder\big(close(X\cup\{m\})\big) \\ &\quad \square say?m:X\cap messages \rightarrow Intruder(X) \end{aligned}$$

This implementation is illustrated by the diagram in Figure 14.

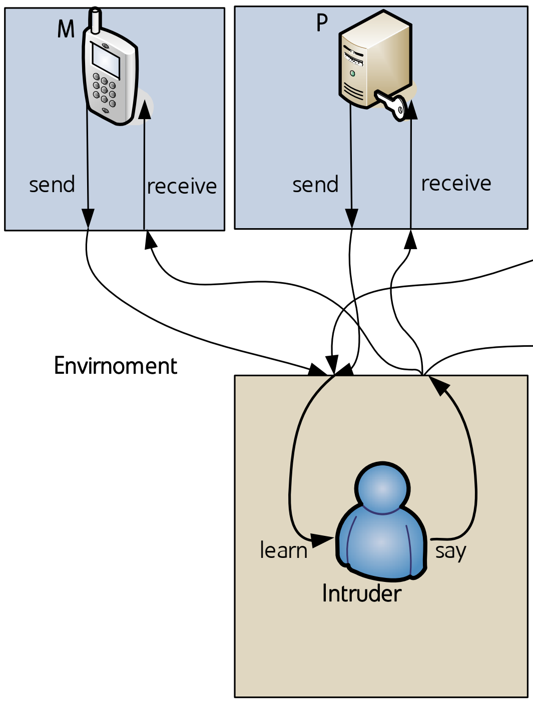


Figure 14: All Network Communication Routed through the Intruder

4.2.1.6 Expressing Protocol Goals

Once the protocol is modeled in the CSP, the final step is to specify all of the goals to be achieved by the protocol or within the protocol run. As introduced in Section 2.4.1.1, the two primary security properties are *secrecy* and *authentication*. It is generally helpful to add signaling channels in the model of the system. These channels do not reflect the agent's behavior, but enable the expression of properties to identify the specific points in the protocol run. These points refer to prior specified properties. Signaling messages consist of a *claim*, a sequence of *users* and a *fact* such as a key, a nonce etc. *Claims* correspond to states of the signaling agent, *users* correspond to the agents the signaling is associated with and finally, *Facts* contain the actual information of a claim.

In order to realize the secrecy statement, a $Claim\_secret$ message is inserted at the end of the protocol-description stating that the intruder is unable to obtain the secret m in the protocol run. The secrecy claim is announced after a full protocol run has taken place. This is illustrated in Figure 15.

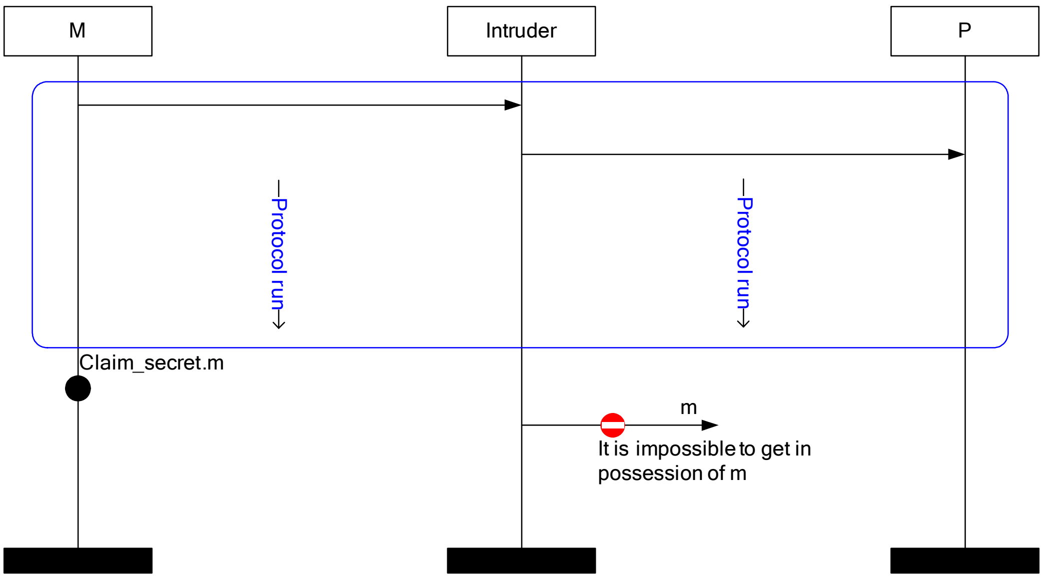


Figure 15: Secrecy Claim

The realization utilizes the CSP description of each agent's role combined with the secret claim. An agent's run is in the form of:

$$
\begin{aligned}
&Initiator(N,na) = \\
&env\,?\,P : Agent \rightarrow send.N.P.N.na \\
&\underset{\substack{K_{SMS} \in Key \\ nb \in Nonce \\ m \in T}}{\square} \left(\begin{array}{c} receive.M.N.\{nb\}_{K_{SMS}}.m \rightarrow \\ send.N.P.m.\{nb\}_{K_{SMS}} \rightarrow signal.Claim\_Secret.N.P.nb \\ \rightarrow Session(N,P,K_{SMS,}na,nb) \end{array} \right)
\end{aligned}
$$

These agents interact with the intruder as:

$$
\begin{aligned}
Intruder(X) = {} & learn\,?\,m : messages \rightarrow Intruder(close(X \cup \{m\})) \\
& \square say\,?\,m : X \cap messages \rightarrow Intruder(X)
\end{aligned}
$$

Therefore, a protocol can be described as a combination of all participating users as in:

$$protocol = Agent_N \;|||\; Agent_P \;|||\; Agent_M$$

while the resulting system looks like:

$$System = protocol \,\|\, Intruder$$

The properties of the security protocol can now be expressed using the trace specifications $S(tr)$. The protocol specification is correct with respect to the intruder and the specification if

$protocol \,\|\, Intruder$ **sat** $S(tr)$

If the modeled system given in Figure 14 is expanded by an additional leak channel, as shown in Figure 16, the secrecy statement can be expressed by:

$$\begin{aligned}&Secret_{N,P}(tr) = \forall m \bullet signal.Claim\_Secret.N.P.m\\&in\ tr \wedge N \in Honest\_Agents \wedge P \in Honest\_Agents\\&\Rightarrow \neg(leak.m\ in\ tr)\end{aligned}$$

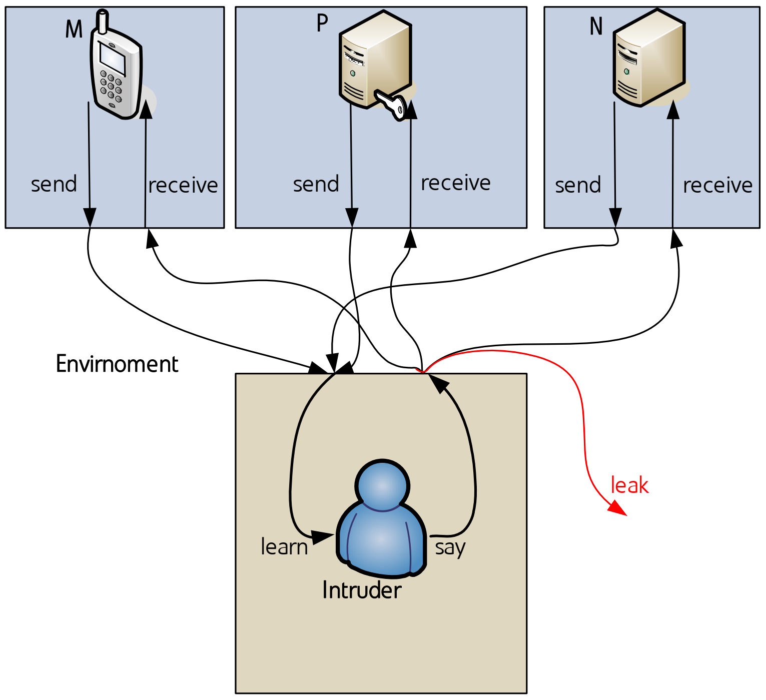


Figure 16: System Including an Additional Leak Channel

The second important property of cryptographic protocols is *Entity Authentication*. This property is also realized by additional signals. The signal *Commit.P.N* event will be implemented into the description of *N's* run of the protocol in order to mark the point where authentication between *P* and *N* has to be achieved. The occurrence of this event is read as:

"Agent N has completed a protocol run apparently with P."

A second signal event *Running.P.N* is read as:

"Agent P is following a protocol run apparently with N."

The *Running* event must always occur before the *Commit* signal is sent (see Figure 17). If the protocol consisted of two nonces and one key, the *Commit* signal would be specified as $Commit.P.N.na.nb.K_{SMS}$, which can be read as:

"Agent P has completed a protocol run apparently with N and with nonces na and nb and with the key K_{SMS}."

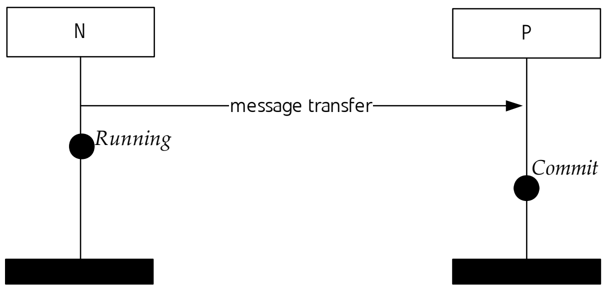


Figure 17: Authentication Events in CSP

Each party will introduce a *Commit* signal (usually) at the end of a protocol run and a *Running* signal to correspond with the other party's *Commit* signal. The signals for the defined roles of *Initiator* and *Responder* will read as $signal.Commit\_Initiator.P.N.na.nb.K_{SMS}$ and analog for *Responder*. The trace specification is relatively simple: The requirement that an event *e* should precede another event *d* is expressed as

$$tr'\,\hat{}\,\langle d\rangle \leq tr \Rightarrow e \;\; in \;\; tr'$$

If $tr'\,\hat{}\,\langle d\rangle$ is a prefix of the trace *tr*, then *e* should appear in tr'. This can be abbreviated using:

e precedes *d*, *d* in $tr \Rightarrow$ *e* in *tr*

and in the case of *Running* and *Commit*:

Running precedes Commit

4.2.2 Failure Divergence Refinement

Failure Divergence Refinement (FDR) [FDR99] is a commercial model checker distributed by *Formal Systems*. This is based on the CSP process algebra. The FDR is capable of checking whether a security model (e.g. a network protocol) satisfies certain security properties such as secrecy, agreements, etcetera, by proving that an *implementation* is a refinement of a corresponding security *specification*. FDR provides trace refinements, failure refinements and failure divergence refinements, by which it is able to detect deadlocks and livelocks. The trace refinement is used for security verification due to the fact that the behavior of a process can be expressed as a trace. This enables the complete reconstruction of the evolution history of a state. In fact the FDR is able to extract attacks and also show an example of the event trace. Therefore, even combinations of attacks such as eavesdropping, interleave, reflection, man-in-the-middle (MIM), etc., as discussed in Section 2.4.3, can be discovered in a single trace. The refinement check is realized by compiling a CSP description of a system into a *Labeled Transition System (LTS)* [Ros82]. The corresponding graph is analyzed in order to prove as to whether a trace exists that has reached a nonconfirming state. All states that are nonconfirming to the specified security properties are referred to as insecure. As a result of this graph-analysis, the expected run time increases exponentially similar to the graph isomorphism problem. This leads to the expectation that the FDR is only capable of handling small protocols with a couple of messages and the implementation of only a few instances. This is expected to be one of the major shortcomings in comparison with other approaches that are described in [CM04]. However, the range of FDR-capability has been increased by using a stringent graph simplification algorithm prior to beginning the actual evaluation.

4.2.2.1 The Casper/FDR Approach

As a result of the complexity of the CSP process algebra definitions and the highly automated process of constructing a CSP code from a *security protocol specification (SPL)*, the easy-to-use SPL-to-CSP-compiler *Casper* was developed by Gavin Lowe [Low97, Low98]. Casper reads `.spl` input-files containing a detailed protocol description and a specification the protocol is verified against. Casper subsequently creates a full CSP process description as a `.csp` output-file which can be verified by the model checker FDR. The structure of the test setup is illustrated in Figure 18. CasperFDR[4] is a Java based graphical frontend that allows easy selection of the files and provides interpretations of the results obtained by the FDR in plaintext.

[4] In the further study, the terms are separated so that Casper can denote the compiler and the script-language of the corresponding input file. The term CasperFDR is used for the graphical frontend and the attack interpretation at a higher level, while the term FDR refers to the model checker and attack traces at the system level. These are actual examples of attacks. The expression Casper/FDR denotes the approaches in general.

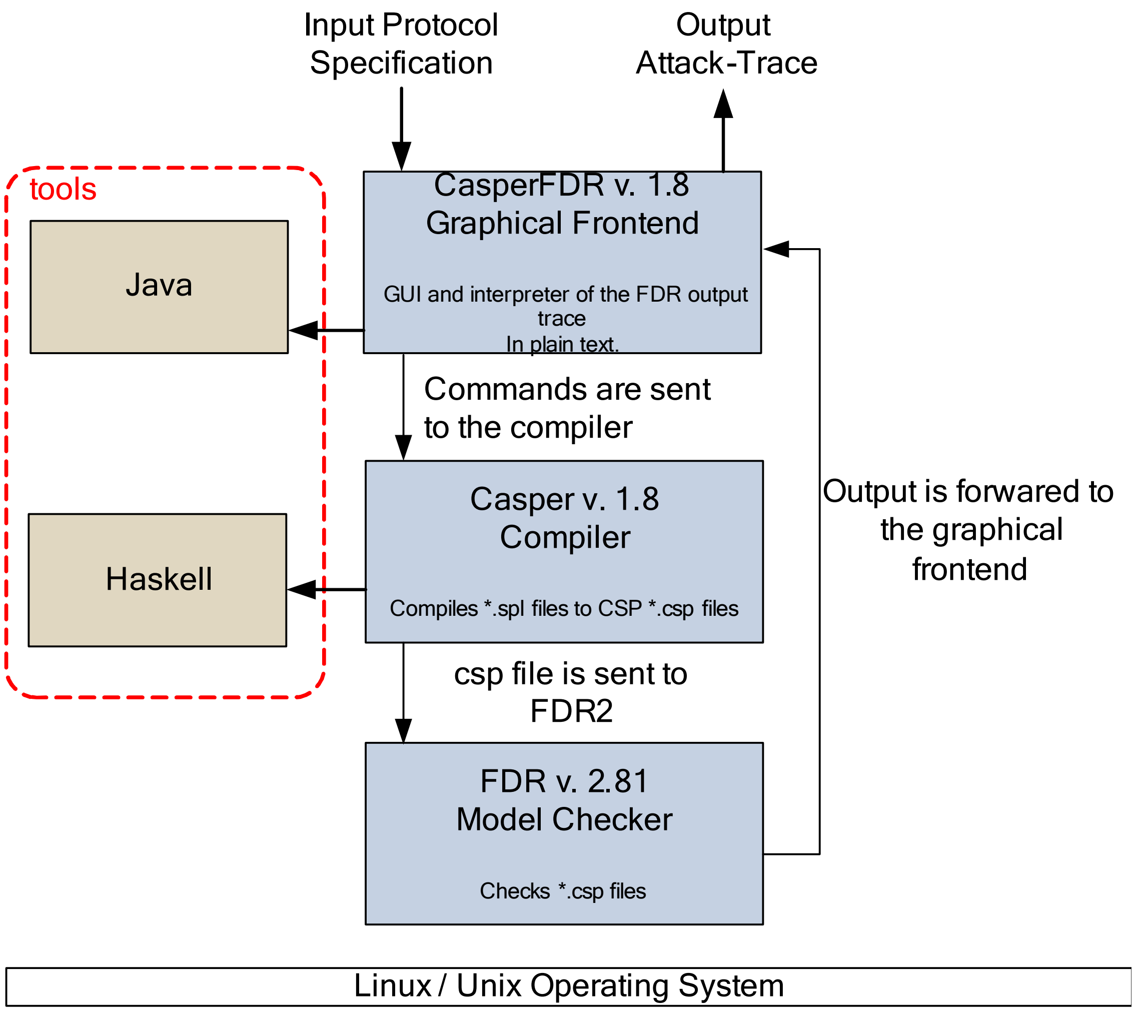

Input Protocol Specification
Output Attack-Trace
tools
Java
Haskell
CasperFDR v. 1.8 Graphical Frontend
GUI and interpreter of the FDR output trace
In plain text.
Commands are sent to the compiler
Casper v. 1.8 Compiler
Compiles *.spl files to CSP *.csp files
csp file is sent to FDR2
FDR v. 2.81 Model Checker
Checks *.csp files
Output is forwarded to the graphical frontend
Linux / Unix Operating System

Figure 18: Casper/FDR System Environment

4.2.3 The Casper Compiler

Casper simplifies CSP-code production by reducing the necessary input to a detailed specification of the protocol. The Casper notation is similar to that which appears in academic literature. In order to express for example, that the *pAR* sends the nonce *na* to the *nAR* and *na* is encrypted by the *nAR*'s public key, one classically writes:

$$pAR \rightarrow nAR : \{na\}_{PK(nAR)}$$

As Casper expects ASCII-text as an input, the message above is expressed so:

P -> N : {na}{PK(N)}

The protocol described in standard notation is considered as a template, parameterized by the *free variables* in it. These free variables correspond to the abstract specification of the protocol without determining the actual values. Thus, these variables are instantiated with *actual variables* in the protocol run. The actual variables are dependent on the actual scenario under which the protocol is examined. Each scenario describes an environment that consists of real nodes and these can vary from a couple of nodes to a potentially unlimited amount of them. However, normally only a few nodes are examined without leaking generality of the consideration. Nevertheless, nodes can adopt several roles within the same scenario enabling the actual variables to instantiate free variables a number of times. This especially occurs in cases of concurrent protocol runs. Hence, the Casper input file requires a specification combining both the abstract protocol and the actual scenario (also referred to as system). Therefore, the Casper input-file has two components:

- A definition of the way in which the protocol operates. The *protocol description section* is combined with a *free variables section.* These both describe messages and the means in which they are exchanged by the different types of agents. They also set out the initial knowledge of all agents, the specification of what the protocol should achieve and a definition of the algebraic equivalences.
- A definition of the actual system that is to be checked. It defines the actual variables and instances of agents taking part in an actual system.

In order to sum up, it can be said that the first part defines the protocol and the second part can be thought of as an image of the protocol function by instantiating the parameters.

4.2.4 Protocol Definition for Casper

The protocol definition employed by Casper, is presented below. It is described in various sections by providing examples from further IDKE analyses, where *P* stands for the *pAR*, *N* for the *nAR* and *M* for the *MN*.

Protocol Description

The protocol is started by an initial message from the environment to an agent, e.g. `-> N : M`, which notifies *N* of *M's* identity in order to enable *N* to start a protocol run with *M*. Thus, a protocol description can appear as :

```
#Protocol description
0.   -> N : M
1. N -> M : N, pkn
2. M -> N : P
```

Free Variables

The section on free variables contains all of the parameters in the protocol, e.g. the agents, nonces, keys etc. An example of simple free variables can be expressed as:

```
#Free variables
M, N, P : Agents
na : Nonce
PK : Agent -> PublicKey
SK : Agent -> SecretKey
```

The last two items define the functions of *PK* and *SK*. These functions when applied to an agent, deliver the corresponding public key (`PublicKey`) or secret key (`SecretKey`). The information also provided in this section covers the definition of inverses
`InverseKeys = (PK, SK)` describes this mathematical property.

Processes

Each running agent in the system is modeled as a CSP process. Therefore, the processes have titles designated to them. These represent the role of the agents as follows:

```
#Processes
INITIATOR(M, na) knows PK, SK(M)
RESPONDER(N) knows PK, SK(N)
```

These titles are defined in the CSP processes by capital letters, whereas the starting knowledge of the corresponding agent is given in brackets. The functional knowledge is added after the keyword `knows`. Hence, the first line reads as:

"The process INITIATOR knows M and na and is able to compute PK on any agent, but it can only compute SK of M."

Specification

In the section on specifications, the security properties are stated:

```
#Specification
Secret(M, knew, [N])
Agreement(M,N,[na])
Agreement(P,N,[ksms])
```

"`Secret`" specifies the secrecy property and reads as:

"M believes that K_{NEW} is only known to him and optionally to N."

This specification will fail, if *A* can complete a run, where the intruder does not legitimately take the role of N, but the intruder leans the value of A gives to s.

Therefore, the intruder must be prevented from obtaining the possession of the variable representing K_{NEW}.

`Agreement (P, N, [ksms])` reads as:

"If P completes a run of the protocol, apparently with N, then N has been running the protocol, apparently with P. The two agents have also agreed upon the roles that each other have taken and also upon the value of the session-key $K_{SMS.}$ This creates a one-to-one relationship between the runs of P and those of N."

The above assertion is the combination of the CSP's *Running* and *Commit* events as described in Section 4.2.1.6.

4.2.4.1 System Definition

The actual system is specified in the second part of the Casper script file. This also contains a number of sections:

Actual Variables

This section specifies all of the actual variables in a scenario (or system) that is intended to be examined. The notation is as follows:

```
#Actual variables
NewAR, PrevAR, MobileNode, Mallory : Agent
Na : Nonce
```

The agents are all implemented as a CSP code and with an additional agent named *Mallory* (the intruder) this has to be instantiated. The remainder of the section is similar to that of the `free variables` section due to the necessity of instantiating protocol parameters.

Functions

The section on functions defines their behavior:

```
#Functions
symbolic PK, SK
```

By using the keyword `symbolic` Casper produces its own value in order to represent the result of functions. Here, for example, the functions are described that generate public keys (`PK`) and secret keys (`SK`).

System

The system definition corresponds to the process definition of the protocol description.

```
#System
INITIATOR(M, na)
RESPONDER(N)
```

4.2.4.2 The Intruder

One of the luxuries of Casper is the simplified implementation of the intruder within a separate section. Thus, to enable one to fully describe this most interesting character of the analysis, only two lines are required in order to specify the intruder's knowledge:

```
#Intruder Information
Intruder = Mallory
IntruderKnowledge = {NewAR, PrevAR, MobileNode, Mallory, PK, SK(Mallory)}
```

The *Intruder = Mallory* line simply gives the intruder a name and the second line specifies the knowledge of the intruder. The inclusion of PK means that the intruder has knowledge of all of the agents' public keys, whereas `SK(Mallory)` limits the knowledge of the secret keys to its own key.

4.2.4.3 Forwarding Messages

It is often necessary to implement the function of forwarding parts of messages to the next agent, e.g. because of the inability to decrypt some information. This can be carried out by Casper using the `%` operator:

```
#Protocol description
0.   -> M : N
1. M -> N : {N, na}{ksms} % var
2. N -> P : var % {N, na}{ksms}
```

This example describes the process of sending an encrypted message from *M* to *P* via *N*. *N* might be unable to decrypt the message because it lacks the key K_{SMS}, but the message can be stored in the variable *var*. The stored content can be forwarded to *P*. However, *P* must be able to decrypt it.

4.3 IDKE Modeled in Casper

Modeling a vast security protocol such as the IDKE protocol, requires a systematic approach to version development. Due to the complexity of the IDKE protocol, it is reasonable to abandon the idea of starting to implement the full IDKE specification, even if the FDR would be able to compute such versions. Apart from the lengthy computation time and the protocol fault diagnostics, message optimization would be tedious. It is recommended instead to initially evaluate a basic version of the IDKE protocol which does not demand all of the required properties. This will allow for fast computation and verification by the refinement checker. The developer is thus able to familiarize himself with the effects (e.g. attacks) of marginal protocol modifications such as the removal or adding of some protocol elements. The continuous minimization prior to implementing new properties to the protocol description, assures optimal usage of computation resources and the avoidance of protocol redundancies. This steady approach to making progress is applied and illustrated in this analysis. However, all of the versions illustrated here consider the same system in that they describe a single instance of each node (MN, pAR and nAR). However, the intruder is allowed to instantiate several nodes in order to break the protocol.

4.3.1 The Basic Version

In order to deduce a minimal, but secure version of the IDKE protocol it is advisable to start with the simplest specification by waiving some of the main features of the IDKE protocol. This will enable one to obtain the basic version and allow for any further development.

The task of this version is to act as a first approach without using DH (see Section 2.4.2.2) for the establishment of a secure tunnel between pAR and nAR. This version also does not aim to provide any freshness guarantee, either on the established key or on the authentication as shown in Protocol 2. The IDKE protocol aims to provide the possibility for a pAR to obtain a new session-key called *KNEW*. However, this is not the intention in this basic version. Nevertheless, the version should be secure. The basic version appears as follows:

1. $\text{nAR} \rightarrow \text{MN} \;:\; \text{nAR-ID, PK(nAR)}$
2. $\text{MN} \rightarrow \text{nAR} \;:\; \text{pAR-ID}, \{\text{PK(nAR), nAR-ID, MN-ID, pAR-ID}\}_{K_{SMS}}$
3. $\text{nAR} \rightarrow \text{pAR} \;:\; \text{nAR-ID}, \{\text{PK(nAR), nAR-ID, MN-ID, pAR-ID}\}_{K_{SMS}}$
4. $\text{pAR} \rightarrow \text{nAR} \;:\; \{K_{SMS}, \text{pAR-ID, MN-ID}\}_{PK(nAR)}$
5. $\text{nAR} \rightarrow \text{MN} \;:\; \{K_{NEW}, \text{nAR-ID}\}_{K_{SMS}}$
6. $\text{MN} \rightarrow \text{nAR} \;:\; \{\text{nAR-ID}\}_{K_{NEW}}$

Protocol 2: The Basic Version (Version 1)

The corresponding Casper implementation of Protocol 2 is made by applying the following code:

```
#Free variables
M, N, P : Agent
pkn, pkp, pkmallory : PublicKey
skn, skp, skmallory : SecretKey
ksms : SessionKey
knew : SessionKey

InverseKeys = (pkn, skn), (ksms, ksms), (pkp, skp), (knew, knew),
(pkmallory, skmallory)

#Protocol description
0.   -> N : M
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M, P}{ksms} % token
3. N -> P : N, token % {pkn, N, M, P}{ksms}
4. P -> N : {ksms, P, M}{pkn}
5. N -> M : {knew, N}{ksms}
6. M -> N : {N}{knew}

#Processes
INITIATOR(N, pkn, skn, knew)
RESPONDER(M, ksms, P)
SERVER(P, ksms, M, pkp, skp, pkn)

#Actual variables
MobileNode, NewAR, PrevAR, Mallory : Agent
```

```
PKN, PKP, PKMALLORY : PublicKey
SKN, SKP, SKMALLORY : SecretKey
KSMS, KNEW : SessionKey
InverseKeys = (KSMS, KSMS), (PKN, SKN), (PKP, SKP), (KNEW, KNEW),
(PKMALLORY, SKMALLORY)

#Specification
Secret(M, knew, [N])
Secret(N, ksms, [N, P])
Agreement(P,N,[ksms])
Agreement(M,N,[knew])

#System
INITIATOR(NewAR, PKN, SKN, KNEW)
RESPONDER(MobileNode, KSMS, PrevAR)
SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {MobileNode, NewAR, Mallory, PKN, PKP, PKMALLORY,
SKMALLORY, PrevAR}
```

As can be seen, the protocol uses the concept of forwarding the token, transmitting K_{SMS} and establishing a new key K_{NEW} between nAR and MN. The authentication of the nAR to the MN is given by the pAR through transmitting K_{SMS}. This is encrypted with the public key of the nAR, PK(nAR) and is only used when the nAR is known to the pAR. In this version, PK(nAR) is in the initial knowledge of the pAR which is expressed by `SERVER(P, ksms, M, pkp, skp, pkn)`. The FDR output shows that the four specifications hold.

- The intruder is unable to get in possession of K_{NEW}.
- The intruder is unable to get in possession of K_{SMS}.
- MN and nAR are authenticated to each other and agree on K_{SMS}.
- pAR and nAR are authenticated to each other and agree on K_{NEW}.

It is important to mention that the pAR is able to know K_{NEW} in this version. Subsequent to testing approximately ten different configurations and removing information from different messages, this appears to be the minimal version. The following section illustrates an attack resulting from a modification to the protocol.

4.3.2 Finding an Attack with Casper/FDR

The basic IDKE protocol has only been minimally modified by removing a single identifier from one message, as shown below:

1. $\text{nAR} \rightarrow \text{MN} \;:\; \text{nAR-ID}, \text{PK}(\text{nAR})$
2. $\text{MN} \rightarrow \text{nAR} \;:\; \text{pAR-ID}, \{\text{PK(nAR)}, \text{nAR-ID}, \text{MN-ID}, \text{pAR-ID}\}_{K_{SMS}}$
3. $\text{nAR} \rightarrow \text{pAR} \;:\; \text{nAR-ID}, \{\text{PK(nAR)}, \text{nAR-ID}, \text{MN-ID}, \text{pAR-ID}\}_{K_{SMS}}$
4. $\text{pAR} \rightarrow \text{nAR} \;:\; \{K_{SMS}, \text{pAR-ID}, \cancel{\text{MN-ID}}\}_{PK(nAR)}$
5. $\text{nAR} \rightarrow \text{MN} \;:\; \{K_{NEW}, \text{nAR-ID}\}_{K_{SMS}}$
6. $\text{MN} \rightarrow \text{nAR} \;:\; \{\text{na}, \text{nAR-ID}\}_{K_{NEW}}$

Protocol 3: Basic Version with removed MN-ID (Insecure)

The corresponding Casper implementation of Protocol 3 is described as:

```
#Protocol description
0.    -> N : M
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M, P}{ksms} % token
3. N -> P : N, token % {pkn, N, M, P}{ksms}
4. P -> N : {ksms, P}{pkn}
5. N -> M : {knew, N}{ksms}
6. M -> N : {na, N}{knew}
```

Only line 4 has been modified from `P -> N : {ksms, P, M}{pkn}` to `P -> N : {ksms, P}{pkn}`. However, the protocol is insecure as the FDR has found a trace of an attack. The attack is interpreted by CasperFDR as stated in Figure 19. The system level trace of the attack discovers the actual vulnerability of the protocol. Casper attaches an "`I`" to indicate that an intruder claims to be a different instance.

```
Initialising; please wait....  Ready.


Casper version 1.8
Parsing...
Type checking...
Consistency checking...
Compiling...
```

```
Writing output...
Output written to /home/IDKE_v1.11h.csp
Done


Starting FDR
Checking /home/IDKE_v1.11h.csp


Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found


Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found


Checking   assertion   AUTH1_M::AuthenticateRESPONDERToINITIATORAgreement_na
[T= AUTH1_M::SYSTEM_1
Attack found:
Top level trace:
  MobileNode believes (s)he is running the protocol, taking role RESPONDER,
with NewAR, using data items Na
  NewAR believes (s)he has completed a run of the protocol, taking role
INITIATOR, with PrevAR, using data items Na
System level:
0.            ->   NewAR    : PrevAR
1.  I_NewAR   -> MobileNode : NewAR, PKN
1.   NewAR    ->  I_PrevAR  : NewAR, PKN
2.  MobileNode  ->    I_NewAR      :  PrevAR,  {PKN,  NewAR,  MobileNode,
PrevAR}{KSMS}
2.    I_PrevAR    ->      NewAR       :  PrevAR,  {PKN,  NewAR,  MobileNode,
PrevAR}{KSMS}
3.  I_NewAR   ->   PrevAR   : NewAR, {PKN, NewAR, MobileNode, PrevAR}{KSMS}
4.   PrevAR   ->  I_NewAR   : {KSMS, PrevAR}{PKN}
3.   NewAR    ->  I_PrevAR  : NewAR, {PKN, NewAR, MobileNode, PrevAR}{KSMS}
4.  I_PrevAR  ->   NewAR    : {KSMS, PrevAR}{PKN}
5.   NewAR    ->  I_PrevAR  : {KNEW, NewAR}{KSMS}
5.  I_NewAR   -> MobileNode : {KNEW, NewAR}{KSMS}
6. MobileNode ->  I_NewAR   : {Na, NewAR}{KNEW}
6.  I_PrevAR  ->   NewAR    : {Na, NewAR}{KNEW}


Checking assertion AUTH2_M::AuthenticateSERVERToINITIATORAgreement_ksms [T=
AUTH2_M::SYSTEM_2
```

```
No attack found


Done
```

Figure 19: CasperFDR Output – Attack on Minimized Version

When leaving out the MN-identifier in line 4, the intruder can break the protocol by combining interleave- and man-in-the-middle-attacks.

The attack is illustrated at a high-level that describes what property has been broken and which undesired state has been finally reached:

```
MobileNode believes (s)he is running the protocol, taking role RESPONDER,
with NewAR, using data items Na

  NewAR believes (s)he has completed a run of the protocol, taking role
INITIATOR, with PrevAR, using data items Na
```

The MN runs the protocol acting as `RESPONDER`, which is an instance of `MobileNode`. The nAR acts as `INITIATOR`, which is an instance of `NewAR`. Casper only gives information on the role of the failed run of the protocol. In the following example, the runs that are needed to break the protocol are marked in different colors. In the runs between the main run the assignments could switch.

In the first two stages of the main run, the intruder takes the role of an "nAR", `I_NewAR and` contacts the MN by sending message 1. It is also assumed that the intruder is aware of the IDKE protocol and has realized that the nAR is offering a service to the MN. This fact has been modeled by the intruder as he possesses the knowledge that an AR exists called `NewAR` with a public key `PKN` (as part of the `IntruderKnowledge` specification in the Casper file).

```
0.             ->  NewAR     : PrevAR

1.  I_NewAR    -> MobileNode : NewAR, PKN
```

In the second run, the Intruder *Mallory* pretends to be the pAR (`I_PrevAR`). Meanwhile, the first protocol run continues and the intruder receives message 1 claiming to be the pAR:

```
1.   NewAR     ->  I_PrevAR  : NewAR, PKN
```

The MN has no reason to be suspicious and answers in accordance with the protocol (message 2). The Intruder, *Mallory*, now possesses a token (the content that authenticates the MN which was sent via the nAR to the pAR) and can continue the second run by sending message 2:

```
2. MobileNode ->  I_NewAR    : PrevAR, {PKN, NewAR, MobileNode,

PrevAR}{KSMS}

2.  I_PrevAR  ->   NewAR     : PrevAR, {PKN, NewAR, MobileNode,

PrevAR}{KSMS}
```

Actually, the token has simply been redirected to `NewAR` by the intruder.

Mallory, also pretending to be a nAR "`I-NewAR`", forwards the token to `PrevAR`, in order to receive an answer from `PrevAR`.

```
3.  I_NewAR   ->   PrevAR   : NewAR, {PKN, NewAR, MobileNode, PrevAR}{KSMS}
4.   PrevAR   ->  I_NewAR   : {KSMS, PrevAR}{PKN}
```

Mallory uses the information received from `PrevAR` since he played this role in the second protocol run. Hence, once he has received message 3 of the second protocol run, he knows the correct answer and can respond appropriately by sending message 4.

Incidentally, this behavior cannot only be attained by starting different runs of the protocol, but by also *reflecting* and *interleaving* messages.

```
3.   NewAR    ->  I_PrevAR  : NewAR, {PKN, NewAR, MobileNode, PrevAR}{KSMS}
4.  I_PrevAR  ->   NewAR    : {KSMS, PrevAR}{PKN}
```

Finally, *Mallory* stores the content of message 5 that he has received in the role of pAR and then forwards it by taking on the role of nAR to `MobileNode` (message 5).

```
5.   NewAR    ->  I_PrevAR  : {KNEW, NewAR}{KSMS}
5.  I_NewAR   -> MobileNode : {KNEW, NewAR}{KSMS}
```

The response of `MobileNode` (message 6) is also then forwarded to `NewAR`.

```
6. MobileNode ->  I_NewAR   : {Na, NewAR}{KNEW}
6.  I_PrevAR  ->   NewAR    : {Na, NewAR}{KNEW}
```

Hence, the communication continues with *Mallory* taking on both the roles of, `I_PrevAR` and the `I_NewAR` by simply forwarding the messages gathered in the different protocol runs, *Mallory* can derange the authentication between MN and nAR. This is possible since MN and nAR are unable to detect that they were both communicating with *Mallory*. Thus, the protocol property of authentication has been broken. However, this attack does not discover any secret, but it could lead to massive security holes, if further communications are based on the trust in the identity of the corresponding parties.

The use of the Casper/FDR approach in order to reduce the possibility of over-engineered protocols, assists in understanding why certain information is necessary. In the case of line 4, `P -> N : {ksms, P, M}{pkn}`, `NewAR` would be able to detect the above mentioned attack since the identifier of the MN is confirmed by the `PrevAR`. The nAR in line 0 is instructed to obtain the key from the pAR and thus, is also able to detect the attack.

4.3.3 Minimization of the Basic Version

By successively removing message-components from the initial version and continuously verifying each protocol-modification with FDR, this creates a reduction process which will lead to a minimal version. This process in fact allows for the exclusion of some components. Hence, the identifier of pAR in message 2, message 3 and message 4 and the identifier of nAR in message 5 can be dispensed with. The resulting basic version then looks as follows:

$$
\begin{array}{lll}
1.\ \text{nAR} \rightarrow \text{MN} & : & \text{nAR-ID, PK(nAR)} \\
2.\ \text{MN} \rightarrow \text{nAR} & : & \text{pAR-ID, } \left\{\text{PK(nAR),nAR-ID, MN-ID, } \cancel{\text{pAR-ID}}\right\}_{K_{SMS}} \\
3.\ \text{nAR} \rightarrow \text{pAR} & : & \text{nAR-ID, } \left\{\text{PK(nAR),nAR-ID, MN-ID, } \cancel{\text{pAR-ID}}\right\}_{K_{SMS}} \\
4.\ \text{pAR} \rightarrow \text{nAR} & : & \left\{K_{SMS}, \cancel{\text{pAR-ID}}\text{, MN-ID}\right\}_{PK(nAR)} \\
5.\ \text{nAR} \rightarrow \text{MN} & : & \left\{K_{NEW}, \cancel{\text{nAR-ID}}\right\}_{K_{SMS}} \\
6.\ \text{MN} \rightarrow \text{nAR} & : & \left\{\text{na, nAR-ID}\right\}_{K_{NEW}}
\end{array}
$$

Protocol 4: Basic Version Light (Secure)

The corresponding Casper protocol description of Protocol 4 is given as.

#Protocol description

```
0.    -> N : M
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M}{ksms} % token
3. N -> P : N, token % {pkn, N, M}{ksms}
4. P -> N : {ksms, M}{pkn}
5. N -> M : {knew}{ksms}
6. M -> N : {na, N}{knew}
```

Detailed information on the analyses and the corresponding CSP files for all Casper input files can be found in the Technical Report [ST05].

4.3.4 Secure Tunnel Implementation

Apart from a secure key transfer between the ARs, it is desirable to have a secure tunnel for this transfer. This tunnel implementation is an intermediate step required for providing forward secrecy. Subsequently, the forward secrecy is only dependent on the used tunnel key. The security of the tunnel should be initially established, while the forward secrecy and freshness requirements are added at a further development stage. These versions are based on a minimized former IDKE version.

In the first version, the tunnel key is not required to provide forward secrecy and thus, is not established with Diffie-Hellman. It is transferred from pAR to nAR based on public keys. The protocol then appears so:

1. $\text{nAR} \rightarrow \text{MN} \ : \ \text{nAR-ID, PK}(\text{nAR})$
2. $\text{MN} \rightarrow \text{nAR} \ : \ \text{pAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{K_{SMS}}$
3. $\text{nAR} \rightarrow \text{pAR} \ : \ \text{nAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{K_{SMS}}$
4. $\text{pAR} \rightarrow \text{nAR} \ : \ \text{pAR-ID, } \{K_{TUNNEL}, \text{na, nAR-ID, MN-ID}\}_{PK(nAR)}$
5. $\text{nAR} \rightarrow \text{pAR} \ : \ \{\text{pAR-ID, nAR-ID, MN-ID, na}\}_{K_{TUNNEL}}$
6. $\text{pAR} \rightarrow \text{nAR} \ : \ \{\text{na, } K_{SMS}, \text{pAR-ID, nAR-ID, MN-ID, PK}(\text{MN})\}_{K_{TUNNEL}}$
7. $\text{nAR} \rightarrow \text{MN} \ : \ \{\{K_{NEW}\}_{SK(nAR)}, \text{MN-ID, nAR-ID}\}_{PK(MN)}$

Protocol 5: Tunnel Version

Protocol 5 implemented in Casper looks as follows:

```
#Free variables
M, N, P : Agent
pkn, pkp, pkm, pkmallory : PublicKey
skn, skp, skm, skmallory : SecretKey
ksms : SessionKey
ktunnel : SessionKey
knew : SessionKey
na, nb : Nonce

InverseKeys = (pkn, skn), (ksms, ksms), (pkp, skp), (knew, knew), (ktunnel,
ktunnel),  (pkm, skm), (pkmallory, skmallory)


#Protocol description
0.   -> N : M
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M}{ksms} % token
3. N -> P : N, token % {pkn, N, M}{ksms}
4. P -> N : P, {ktunnel, na, N, M}{pkn}
5. N -> P : {P, N, M, na}{ktunnel}
6. P -> N : {na, ksms, P, N, M, pkm}{ktunnel}
7. N -> M : {{knew}{skn}, M, N}{pkm}
```

```
#Processes
INITIATOR(N, pkn, skn, knew)
RESPONDER(M, ksms, P, pkm, skm)
SERVER(P,ksms, M, pkp, skp, pkn, na, ktunnel, pkm)

#Actual variables
MobileNode, NewAR, PrevAR, Mallory : Agent
PKN, PKP, PKM, PKMALLORY : PublicKey
SKN, SKP, SKM, SKMALLORY : SecretKey
KSMS, KNEW, KTUNNEL : SessionKey
Na : Nonce
InverseKeys = (KSMS, KSMS), (PKN, SKN), (PKP, SKP), (KNEW, KNEW), (KTUNNEL,
KTUNNEL),(PKM, SKM), (PKMALLORY, SKMALLORY)

#Specification
Secret(P, ktunnel, [N])
Secret(N, knew, [M])
Secret(M, ksms, [N])
Secret(P, na, [N])
Agreement(P, N, [ktunnel])
Agreement(N, M, [knew])

#System
INITIATOR(NewAR, PKN, SKN, KNEW)
RESPONDER(MobileNode, KSMS, PrevAR, PKM, SKM)
SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN, Na, KTUNNEL, PKM)

#Intruder Information
Intruder = Mallory
IntruderKnowledge  =  {MobileNode,  NewAR,  Mallory,  PKN,  PKP,  PrevAR,
PKMALLORY, SKMALLORY}
```

The FDR output has stated that this protocol is secure. However, this protocol version contains some over-engineered lines. This is a safeguard which is adopted as a means of security. Minimization is subsequently carried out in Section 4.3.6.

4.3.5 Tunnel Freshness Removal

It is important to be aware of the fact that the Casper/FDR approach is unable to check “real” freshness attacks. Therefore, the freshness property cannot be validated, but needs to be carefully considered when implementing the protocol. Hence, session identifiers or timestamps will be required to be attached to each message. In the following, the freshness guaranteeing elements have been removed for performance reasons.

The implementation is based on the version introduced in Section 4.3.4 where message 4, 5 and 6 did not contain the nonce, but where message 7 was given an additional authentication by means of the encryption with K_{SMS}. The protocol description is as follows:

$$
\begin{array}{lll}
1.\ \text{nAR} \rightarrow \text{MN} & : & \text{nAR-ID, PK}(\text{nAR}) \\
2.\ \text{MN} \rightarrow \text{nAR} & : & \text{pAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{K_{SMS}} \\
3.\ \text{nAR} \rightarrow \text{pAR} & : & \text{nAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{K_{SMS}} \\
4.\ \text{pAR} \rightarrow \text{nAR} & : & \text{pAR-ID, } \{K_{TUNNEL}, \text{nAR-ID, MN-ID}\}_{PK(nAR)} \\
5.\ \text{nAR} \rightarrow \text{pAR} & : & \{\text{pAR-ID, nAR-ID, MN-ID}\}_{K_{TUNNEL}} \\
6.\ \text{pAR} \rightarrow \text{nAR} & : & \{K_{SMS}, \text{pAR-ID, nAR-ID, MN-ID, PK}(\text{MN})\}_{K_{TUNNEL}} \\
7.\ \text{nAR} \rightarrow \text{MN} & : & \left\{\{K_{NEW}\}_{SK(nAR)}, \text{MN-ID, } \{\text{nAR-ID}\}_{K_{SMS}}\right\}_{PK(MN)}
\end{array}
$$

Protocol 6: Tunnel Version without Nonces (Secure)

The Casper script file of Protocol 6 contains the following protocol description:

```
#Protocol description
0.   -> N : M
[N!=M]
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M}{ksms} % token
3. N -> P : N, token % {pkn, N, M}{ksms}
4. P -> N : P, {ktunnel, N, M}{pkn}
5. N -> P : {P, N, M}{ktunnel}
6. P -> N : {ksms, P, N, M, pkm}{ktunnel}
7. N -> M : {{knew}{skn}, M, {N}{ksms}}{pkm}
```

The expression `[N!=M]` is also employed for performance reason and advises Casper to skip all cases in which the MN adopts the role of the nAR. Such cases are meaningless, since the MN already has knowledge of the session-key K_{SMS} and thus, would not perform such a request.

The verification by FDR could not discover any attack so that CasperFDR stated this version also as secure by the output "No attack found" for all specified properties.

4.3.6 Complexity Reduction by Tunnel Minimization

It was also necessary to reduce the complexity of the protocol since the limit of FDR would be exceeded by adding more messages of encryption to the protocol. The aim is to reduce the complexity of the tunnel by deleting some tunneled elements. However, nonces are still included in this version. The resulting minimized version is still based on Protocol 6. Any changes are depicted by the following colors (red represents what has been removed and blue those that have been modified):

1. $\text{nAR} \rightarrow \text{MN} \ : \ \text{nAR-ID}, \text{PK}(\text{nAR})$
2. $\text{MN} \rightarrow \text{nAR} \ : \ \text{pAR-ID}, \{\text{PK}(\text{nAR}), \text{nAR-ID}, \text{MN-ID}\}_{K_{SMS}}$
3. $\text{nAR} \rightarrow \text{pAR} \ : \ \text{nAR-ID}, \{\text{PK}(\text{nAR}), \text{nAR-ID}, \text{MN-ID}\}_{K_{SMS}}$
4. $\text{pAR} \rightarrow \text{nAR} \ : \ \text{pAR-ID}, \{K_{TUNNEL}, \text{na}, \text{nAR-ID}, \text{MN-ID}\}_{PK(nAR)}$
5. $\text{nAR} \rightarrow \text{pAR} \ : \ \{\text{pAR-ID}, \text{nAR-ID}, \text{MN-ID}, \text{na}\}_{K_{TUNNEL}}$
6. $\text{pAR} \rightarrow \text{nAR} \ : \ \{\text{na}, K_{SMS}, \text{pAR-ID}, \cancel{\text{nAR-ID}}, \text{MN-ID}, \text{PK}(\text{MN})\}_{K_{TUNNEL}}$
7. $\text{nAR} \rightarrow \text{MN} \ : \ \{\{K_{NEW}\}_{SK(nAR)}, \cancel{\text{MN-ID}}, \text{nAR-ID}\}_{PK(MN)}$

Protocol 7: Tunnel Minimal Version (Secure)

The corresponding Casper script file of Protocol 7 contains the following:

#Protocol description

```
0.    -> N : M
1. N -> M : N, pkn
2. M -> N : P, {pkn,N, M}{ksms} % token
3. N -> P : N, token % {pkn, N, M}{ksms}
4. P -> N : P, {ktunnel, na, N, M}{pkn}
5. N -> P : {P, N, M, na}{ktunnel}
6. P -> N : {na, ksms, P, M, pkm}{ktunnel}
7. N -> M : {{knew}{skn}, N}{pkm}
```

#Specification

```
Secret(P, ktunnel, [N])
Secret(N, knew, [M])
Secret(M, ksms, [N])
Secret(P, na, [N])
Agreement(P, N, [ktunnel])
Agreement(N, M, [knew])
```

According to the specification, FDR has stated that this version is secure. The fulfilled properties are the secrecy of K_{TUNNEL} and the nonce *na* (corresponding to any further confidential data) between pAR and nAR. The authentication is achieved by the agreement of K_{TUNNEL} between pAR and nAR and K_{NEW} between nAR and MN.

4.3.7 Forward Secrecy by Diffie-Hellman

This version is the first approach to establishing K_{TUNNEL_DH}, which is a session-key between nAR and pAR, utilizing the *Diffie-Hellman Key Exchange (DH)*. The first version used DH for both the establishment of K_{TUNNEL} and of K_{NEW} and this proved to be too large for the FDR to handle. The system produced more than 2.000.000 states and the memory resources were inefficient. It is not possible to have a maximal run time of an *NP*-complete graph algorithm with 2.000.000 states. A termination within an appropriate amount of time cannot be guaranteed and it is most unlikely that this can be achieved. Therefore, the approach was modified in order to only support K_{TUNNEL_DH} as a basis for further minimization.

1. $\text{nAR} \rightarrow \text{MN} \ : \ \text{nAR-ID, PK}(\text{nAR})$
2. $\text{MN} \rightarrow \text{nAR} \ : \ \text{pAR-ID}, \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID, na}\}_{\text{K}_{\text{SMS}}}$
3. $\text{nAR} \rightarrow \text{pAR} \ : \ \text{nAR-ID}, \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID, na}\}_{\text{K}_{\text{SMS}}}$
4. $\text{pAR} \rightarrow \text{nAR} \ : \ \text{pAR-ID}, \{HK_A := Exp(Gen, x), \text{na, MN-ID, PK}(\text{pAR})\}_{\text{PK(nAR)}}$
5. $\text{nAR} \rightarrow \text{pAR} \ : \ \left\{\text{na}, \{HK_B := Exp(Gen, y)\}_{\text{SK(nAR)}}\right\}_{\text{PK(pAR)}}$

$\langle K_{TUNNEL} := Exp(HK_B, x) = Exp(HK_A, y)\rangle$

6. $\text{pAR} \rightarrow \text{nAR} \ : \ \{\text{na}, \text{K}_{\text{SMS}}, \text{PK}(\text{MN})\}_{\text{K}_{\text{TUNNEL}}}$
7. $\text{nAR} \rightarrow \text{MN} \ : \ \left\{\{\text{K}_{\text{NEW}}, \text{na}\}_{\text{SK(nAR)}}, \{\text{nAR-ID}\}_{\text{K}_{\text{SMS}}}\right\}_{\text{PK(MN)}}$

Protocol 8: Tunnel Version with DH (Insecure)

The implementation of a version incorporating DH is much more complex than the previous protocols. In Protocol 8 the messages 4 and 5 represent DH. The corresponding Casper script file will be explained in more detail later:

```
#Free variables
datatype Field = Gen | Exp(Field,Num) unwinding 2
halfkeyA, halfkeyB, ktunnel : Field
x, y : Num

M, N, P : Agent
pkn, pkp, pkm, pkmallory : PublicKey
skn, skp, skm, skmallory : SecretKey
ksms, knew : SessionKey
na : Nonce

InverseKeys = (pkn, skn), (ksms, ksms), (pkp, skp), (knew, knew), (ktunnel,
ktunnel), (pkm, skm), (pkmallory, skmallory), (Exp, Exp), (Gen, Gen)
```

As can be seen, the first three lines define the DH operation. The exponential function expects a value of type `Field` (a half-key or *KTUNNEL*) and produces another `Field` by calculating DH.

```
#Protocol description
0.   -> N : M
1. N -> M : N, pkn
[N!=M]
2. M -> N : P, {pkn,N, M, na}{ksms} % token
[M!=N]
3. N -> P : N, token % {pkn, N, M, na}{ksms}
[N!=P]
4. P -> N : P, {Exp(Gen,x) % halfkeyA, na, M, pkp}{pkn}
[P!=M]
< ktunnel := Exp(halfkeyA, y) >
5. N -> P : {na, {Exp(Gen,y) % halfkeyB}{skn}}{pkp}
[N!=P]
< ktunnel := Exp(halfkeyB, x) >
6. P -> N : {na, ksms, pkm}{ktunnel}
[P!=N]
7. N -> M : {{knew, na}{skn}, {N}{ksms}}{pkm}
```

In order to save the computation resources, all of the protocol stages have been modified to only being able to continue when the sender is unequal to the receiver. The session-key has been calculated as `< ktunnel := Exp(halfkeyA, y) >`. This is then known to the receiver of the above message.

```
#Equivalences
forall x, y : Num . Exp ( Exp(Gen,x), y ) = Exp( Exp(Gen,y), x )
```

By using this equivalence-specification, the properties of DH are adapted to the CSP environment. It is most important to have the abstract function instead of an exact mathematical formalism. It is essential to recognize this as it detaches the IDKE from the DH. The abstracted version can in any case only be realized by mathematical formalism.

```
#Processes
INITIATOR(N, pkn, skn, knew, y)
RESPONDER(M, ksms, P, pkm, skm, na)
SERVER(P,ksms, M, pkp, skp, pkn, pkm, x)

#Actual variables
MobileNode, NewAR, PrevAR, Mallory : Agent
PKN, PKP, PKM, PKMALLORY : PublicKey
SKN, SKP, SKM, SKMALLORY : SecretKey
KSMS, KNEW : SessionKey
X, Y, Z : Num
Na : Nonce
InverseKeys = (KSMS, KSMS), (PKN, SKN), (PKP, SKP), (PKM, SKM), (PKMALLORY,
SKMALLORY), (KNEW, KNEW)

#Specification
Secret(P, ktunnel, [N])
Secret(N, ktunnel, [P])
Secret(N, knew, [M])
Secret(M, knew, [N])
Secret(M, ksms, [N])
Secret(N, ksms, [M])
Agreement(P, N, [ktunnel])
Agreement(N, M, [knew])
Agreement(P, N, [ksms])
```

```
#System
INITIATOR(NewAR, PKN, SKN, KNEW, Y)
RESPONDER(MobileNode, KSMS, PrevAR, PKM, SKM, Na)
SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN, PKM, X)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {MobileNode, NewAR, Mallory, PKN, PKP, PrevAR,
PKMALLORY, SKMALLORY, Z}
```

The remaining implementation is self- explanatory. The specifications are as general as possible. The FDR is able to verify this huge protocol by being run on a fast workstation with 6GB RAM and is capable of completing the computation in about 5 hours. However, FDR stated that the protocol was insecure. The verification showed that both of the agreements on *K_{NEW}* between nAR and MN had failed. The CasperFDR output containing the FDR attack trace is illustrated in Figure 20 and Figure 21.

```
 NewAR believes (s)he has completed a run of the protocol, taking role
INITIATOR, with PrevAR, using data items Exp__, (Exp__, (Gen__, Y), X)


0.              ->    NewAR     : MobileNode
1.    NewAR     -> I_MobileNode : NewAR, PKN
1.  I_Mallory   ->  MobileNode  : Mallory, PKN
2.   MobileNode   ->   I_Mallory    : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
2. I_MobileNode ->      NewAR       : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
3.  I_Mallory    ->      PrevAR     : Mallory, {PKN, Mallory, MobileNode,
Na}{KSMS}
4.     PrevAR     ->   I_Mallory    : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
3.      NewAR       ->     I_PrevAR     : NewAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
4.    I_PrevAR    ->     NewAR       : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
5.    NewAR     ->   I_PrevAR   : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
5.  I_Mallory   ->    PrevAR    : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
6.    PrevAR    ->  I_Mallory   : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
6.   I_PrevAR   ->    NewAR     : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
7.    NewAR     -> I_MobileNode : {{KNEW, Na}{SKN}, {NewAR}{KSMS}}{PKM}
```

Figure 20: CasperFDR Output – Attack 1 on Tunnel Version (Protocol 8)

```
  NewAR believes (s)he has completed a run of the protocol, taking role
INITIATOR, with PrevAR, using data items KSMS


0.                -> NewAR          : MobileNode
1.    NewAR       -> I_MobileNode   : NewAR, PKN
1.  I_Mallory     -> MobileNode     : Mallory, PKN
2.  MobileNode    -> I_Mallory      : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
2. I_MobileNode   -> NewAR          : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
3.  I_Mallory     -> PrevAR         : Mallory, {PKN, Mallory, MobileNode,
Na}{KSMS}
4.    PrevAR      -> I_Mallory      : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
3.     NewAR      -> I_PrevAR       : NewAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
4.   I_PrevAR     -> NewAR          : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
5.    NewAR       -> I_PrevAR       : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
5.  I_Mallory     -> PrevAR         : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
6.    PrevAR      -> I_Mallory      : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
6.   I_PrevAR     -> NewAR          : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
7.    NewAR       -> I_MobileNode   : {{KNEW, Na}{SKN}, {NewAR}{KSMS}}{PKM}
```

Figure 21: CasperFDR Output – Attack 2 on Tunnel Version (Protocol 8)

By analyzing the attack in more detail it reveals the discovery of a complex interleave-attack. An interpretation of the attack is given inline:

```
NewAR believes (s)he has completed a run of the protocol, taking role
INITIATOR, with PrevAR, using data items Exp__, (Exp__, (Gen__, Y), X)
```

The run consists of two interleaved protocols that are distinguished here in colored lettering.

```
0.                -> NewAR          : MobileNode
1.    NewAR       -> I_MobileNode   : NewAR, PKN
```

The intruder claiming to be the MN (`I_MobileNode`) captures message 1 in order to establish a connection to the nAR. Then, the intruder changes the MN's identifier and inserts its own before sending message 1 to the MN. Therefore, the MN replies by sending message 2 directly to the intruder.

```
1.  I_Mallory     -> MobileNode     : Mallory, PKN
2.  MobileNode    -> I_Mallory      : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
```

The intruder acts as the man in the middle and forwards the content of message 2 to the nAR by sending message 2 from a different protocol run.

```
2.  I_MobileNode  ->      NewAR        : PrevAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
```

The intruder also utilizes the same content and sends it in message 3 to the pAR claiming to be the nAR. It should be mentioned that the content of message 2 and message 3 is exactly the same and contains in both cases the intruder's identity: *Mallory*.

```
3.   I_Mallory    ->      PrevAR      : Mallory, {PKN, Mallory, MobileNode,
Na}{KSMS}
```

The pAR responds to the intruder by message 4:

```
4.     PrevAR     ->   I_Mallory    : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
```

Once *Mallory* has received message 4, he waits for the nAR to send message 3 and then answers by sending message 4, the content of which he has gained from message 4.

```
3.      NewAR        ->     I_PrevAR     : NewAR, {PKN, Mallory, MobileNode,
Na}{KSMS}
```

```
4.    I_PrevAR    ->      NewAR        : PrevAR, {Exp(Gen, X), Na, MobileNode,
PKP}{PKN}
```

As the answer is correct, the nAR will respond by sending message 5 to the intruder:

```
5.     NewAR     ->    I_PrevAR    : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
```

The intruder then uses the content of message 5 to send its own message 5 to the pAR:

```
5.   I_Mallory    ->     PrevAR     : {Na, {Exp(Gen, Y)}{SKN}}{PKP}
```

The pAR also sends message 6 back to the intruder:

```
6.     PrevAR     ->   I_Mallory    : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
```

Mallory then forwards the content of message 6 to the nAR by sending message 6 in order to receive message 7 and hence it is authenticated as MN.

```
6.    I_PrevAR    ->     NewAR      : {Na, KSMS, PKM}{Exp(Exp(Gen, X), Y)}
7.     NewAR     -> I_MobileNode : {{KNEW, Na}{SKN}, {NewAR}{KSMS}}{PKM}
```

This attack is similar to that described in Section 4.3.1. It discovers the major vulnerability of the IDKE protocol. Therefore, the authentication has to be improved between the pAR and nAR..

4.3.8 Attack Solution and Final Version

The aim of the "final version" is to withstand attacks, especially those that have been discovered previously. Therefore, the authentication will be improved by a version which can correspond to a combination of the versions presented in Section 4.3.6 and 4.3.7.

This leads to the concept of securing the tunnel establishment by using a session-key sent from pAR to nAR in order to authenticate the two ARs with each other. The tunnel is also used to transfer the DH half-keys between both of the ARs. Thus, enabling the forward secrecy to be provided for the secure key establishment as illustrated in Protocol 9.

1. $\text{nAR} \rightarrow \text{MN} \;:\; \text{nAR-ID, PK}(\text{nAR})$
2. $\text{MN} \rightarrow \text{nAR} \;:\; \text{pAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{\text{K}_{\text{SMS}}}$
3. $\text{nAR} \rightarrow \text{pAR} \;:\; \text{nAR-ID, } \{\text{PK}(\text{nAR}), \text{nAR-ID, MN-ID}\}_{\text{K}_{\text{SMS}}}$
4. $\text{pAR} \rightarrow \text{nAR} \;:\; \text{pAR-ID, } \{\text{K}_{\text{TUNNEL}}, \text{pAR-ID, nAR-ID}\}_{\text{PK}(\text{nAR})}$
5. $\text{nAR} \rightarrow \text{pAR} \;:\; \{\text{nAR-ID}\}_{\text{K}_{\text{TUNNEL}}}$
6. $\text{pAR} \rightarrow \text{nAR} \;:\; \{HK_A := Exp(Gen, x)\}_{\text{K}_{\text{TUNNEL}}}$
7. $\text{nAR} \rightarrow \text{pAR} \;:\; \{HK_B := Exp(Gen, y)\}_{\text{K}_{\text{TUNNEL}}}$

$\langle K_{TUNNEL\_DH} := Exp(HK_B, x) = Exp(HK_A, y) \rangle$

8. $\text{pAR} \rightarrow \text{nAR} \;:\; \{\text{K}_{\text{SMS}}, \text{nAR-ID}\}_{\text{K}_{\text{TUNNEL\_DH}}}$
9. $\text{nAR} \rightarrow \text{MN} \;:\; \{\text{na, nAR-ID, MN-ID}\}_{\text{K}_{\text{SMS}}}$
10. $\text{MN} \rightarrow \text{nAR} \;:\; \{\text{na, MN-ID, nAR-ID}\}_{\text{K}_{\text{SMS}}}$

Protocol 9: IDKE Final Version

The corresponding Casper implementation of Protocol 9 is as follows (the purposes of individual messages and some design choices are explained inline):

```
#Free variables
datatype Field = Gen | Exp(Field,Num) unwinding 2
halfkeyA, halfkeyB, ktunnelDH : Field


M, N, P : Agent
pkn, pkp : PublicKey
skn, skp : SecretKey
ksms, ktunnel : SessionKey
na : Nonce
```

```
x, y : Num


InverseKeys = (pkn, skn), (ksms, ksms), (pkp, skp),(ktunnel, ktunnel),
(Exp, Exp), (Gen, Gen), (ktunnelDH, ktunnelDH)
```

As in prior protocol versions, the protocol parameters referred to as free variables, are used to prepare the roles of MN, nAR and pAR as well as their public/private key pairs. The analysis has shown that all of the specifications can be validated without transferring the MN's public key. Therefore, this has been omitted in order to reduce protocol complexity.

```
#Protocol description
0.   -> N : M
1. N -> M : N, pkn
[N!=M]
2. M -> N : P, {pkn,N, M}{ksms} % token
[M!=N]
3. N -> P : N, token % {pkn, N, M}{ksms}
[N!=P]
4. P -> N : P, {ktunnel, P, N}{pkn}
[P!=N]
5. N -> P : {N}{ktunnel}
[N!=P]
```

At this point, nAR and pAR have agreed on *K_{TUNNEL}* and continue to use it in order to establish forward secrecy for tunnel key *K_{TUNNEL_DH}*.

```
6. P -> N : {Exp(Gen,x) % halfkeyA}{ktunnel}
[P!=N]
< ktunnelDH := Exp(halfkeyA, y) >
7. N -> P : {Exp(Gen,y) % halfkeyB}{ktunnel}
[N!=P]
< ktunnelDH := Exp(halfkeyB, x) >
```

After exchanging the DH-half-keys, both ARs agree on the computed DH-tunnel-key K_{TUNNEL_DH}.

```
8. P -> N : {ksms, N}{ktunnelDH}
[P!=N]
9. N -> M : {na, N, M}{ksms}
[N!=M]
10. M -> N : {na, M, N}{ksms}
```

In the analyzed protocol run, it does not compute any fresh K_{NEW}, but uses K_{SMS} directly. The reason for this is to reduce the problem of the FDR state space exceeding its memory. One of the FDR subprocesses, namely *state 2,* is limited to 3.1 GB of RAM.

```
#Equivalences
forall x, y : Num . Exp ( Exp(Gen,x), y ) = Exp( Exp(Gen,y), x )
```

As previously deduced, the equivalences describe the detachment of the IDKE protocol from DH. Thus, it is sufficient to realize this abstracted expression by a mathematical formalism.

```
#Processes
INITIATOR(N, pkn, skn, y, pkp, na)
RESPONDER(M, ksms, P)
SERVER(P,ksms, M, pkp, skp, pkn, ktunnel, x)
```

```
#Actual variables
MobileNode, NewAR, PrevAR, Mallory : Agent
PKN, PKP, PKMALLORY : PublicKey
SKN, SKP, SKMALLORY : SecretKey
KSMS, KTUNNEL : SessionKey
X, Y, Z : Num
Na, NInt : Nonce

InverseKeys = (KSMS, KSMS), (PKN, SKN), (PKP, SKP), (KTUNNEL, KTUNNEL),
(PKMALLORY, SKMALLORY)
```

The actual variables instantiate the free variables and realize the presence of the intruder. The intruder is equipped with its own nonces and numeric values in order to interfere with the DH.

```
#Specification
Secret(P, ktunnelDH, [N])
Secret(N, ktunnelDH, [P])
Secret(P, ktunnel, [N])
Secret(N, ktunnel, [P])
Secret(M, ksms, [N])
Secret(N, ksms, [M])
Secret(M, na, [N])
Secret(N, na, [M])
Agreement(M, N, [na])
Agreement(N, M, [ksms])
Agreement(P, N, [ktunnel])
Agreement(P, N, [ksms])
Agreement(P, N, [ktunnelDH])
```

The specification describes the properties to be verified by the FDR.

```
#System
INITIATOR(NewAR, PKN, SKN, Y, PKP, Na)
RESPONDER(MobileNode, KSMS, PrevAR)
SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN, KTUNNEL, X)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {MobileNode, NewAR, Mallory, PKN, PKP, PrevAR,
PKMALLORY, SKMALLORY, Z, NInt}
```

CasperFDR states that the above description is secure against the specification as FDR has not found any attack trace. Thus, the analyses have shown that all properties can be verified successfully. The FDR analyses verified:

- The secrecy of K_{TUNNEL}, K_{SMS}, K_{TUNNEL_DH} and even the secrecy of the nonce *na*.
- MN and nAR agree on K_{SMS} and *na*.
- The two ARs agree on K_{TUNNEL}, K_{SMS} and K_{TUNNEL_DH}.

Therefore, the aim of a secure key transfer and forward secrecy has been accomplished. However, due to the complexity, the generation of K_{NEW} has been reduced in message 9 and 10. As MN and nAR have been authenticated with each other, a second DH key exchange is possible in order to provide a new session-key between MN and nAR as well as a forward secrecy for the new session-key.

4.4 Summary, Discussion & Outlook

4.4.1 Summary

This chapter has analyzed the security of the IDKE protocol. The main tasks for verification are:

- Evaluation of the tools for formal security verification.
- Identification of potential attacks on the IDKE protocol and improvements for countering the attacks that are discovered.
- Optimization of the protocol by removing unnecessary parts.
- Verification that all of the requirements have been fulfilled in the final version.
- Determine the limitations on the formal security verification of the chosen model checking approach.

The objective of this analysis is to formally prove that the IDKE protocol is able to fulfill its security properties. However, it is generally difficult to show protocol or program correctness. This is one of the undecidable questions in computer science [AHU74].

4.4.2 Discussion

4.4.2.1 Tool Evaluation

Theorem proving approaches aim to provide such formal proof. Initially, it was decided to verify the IDKE protocol's correctness by either using the BAN logic (see Section 4.1.2.1) or Isabelle (see Section 4.1.2.3). However, both have serious drawbacks in discovering actual attacks since they e.g. do not show any attack trace. Hence, once a protocol cannot be verified, one does not have the possibility of rectifying the problem when the vulnerability is not obvious. BAN logic also has serious disadvantages due to its unrealistically simplified assumptions on authentication. In fact protocols that have been stated as secure by the BAN logic have been discovered to be vulnerable to attacks that break the authentication capabilities.

Therefore, the active search for attacks becomes an essential part of the security verification. As the NPA (see Section 4.1.2.2), which is a hybrid approach for model checking and theorem proving is not publicly available; the Casper/FDR approach was selected. This is currently the most promising approach, especially for identifying attacks on protocols, but it is limited in its capability of verifying huge protocols. The IDKE protocol is in fact immense in size and exceeds the state space of the FDR when one examines the entire protocol without applying any modifications. This results in two conflicting goals: Firstly, of reducing the state space in order to make the FDR capable of analyzing the IDKE protocol; and secondly to have the most general scenario for ensuring that all potential attacks are discovered. Thus, instead of analyzing the protocol in one-piece, the IDKE protocol is successively assembled in versions, verified, minimized and consecutively improved.

4.4.2.2 The Deployment Process

The protocol was successfully improved by beginning with a basic version which aimed to provide only a subset of the desired properties. This version focuses on providing a simple key forwarding and does not guarantee any key forward secrecy or offer any secure tunnel between the ARs. However, under the specified security properties, this version is stated as secure. Nevertheless, the version was not minimal since it contained some unnecessary non-essential parts, such as some IDs. In order to reduce complexity, a number of identifiers within some of the messages were removed. Further analyses demonstrated that some of the identifiers were not required, while others were relevant. Moreover, a complex attack was unexpectedly discovered when removing the MN's identifier in message 4. This attack illustrated that the protocol could be broken by initially interleaving several protocol runs and also by acting as *man in the middle.* As a result of this, the intruder was able to produce an invalid authentication. However, authentication is also an obligatory property of the IDKE protocol, even if no secret information is discovered.

In order to achieve a minimal basic version, the protocol was subsequently trimmed down by exclusively removing unessential parts. This process resulted in a minimal and secure basic version. This version later acted as the basis for producing a final version that contains all of the desired properties.

A secure tunnel was also established and added between the ARs and the resulting version was successfully verified as being secure. Nevertheless, a desired improvement was to add forward secrecy to the tunnel key. Therefore, the version was modified to include a Diffie-Hellman key negotiation mechanism.

However, the integrated DH tunnel was stated as being insecure. The FDR again discovered a complex interleave-attack which resulted in a failed authentication, despite the fact that the key affords forward secrecy. This circumstance led to the concept of adding interleaved tunnels in order to confront the attack. The first secure stated tunnel acts as an auxiliary tunnel for setting up the DH tunnel key. Thus, the final version implements an authenticated tunnel which provides forward secrecy between the ARs.

4.4.2.3 The Final Version

The FDR trace refinement check of the final CSP input script indicates that this version of the IDKE protocol is secure as an attack has not been discovered. However, the Casper/FDR approach is limited in its capabilities of model checking. Depending on the approach chosen, the security analyses will still reveal attacks and if used improperly, the results are unreliable and almost worthless. One main problem is the mapping of reality into the checking environment. The environment in the CSP and the intruder can be modeled to map actual attacks. This means that the modeling of the environment in this way can map such attacks as those on the modifying of messages, the reflecting of messages and so on. Therefore, the CSP-code producer has a direct control over whether, for example, a message can be delayed or not (cable vs. wireless environment). However, major problems arise with attacks on the decryption of electromagnetic radiation, etc. Such attacks are difficult to carry out and are not really advisable for implementation in the formal analysis due to the resulting complexity. Furthermore, it is a profound mistake to assume the security of an implemented protocol purely by its formal analysis. One should always take into account under what assumptions the protocol was actually stated as being secure. The disregard of initial formal assumptions can result in implementation failures that could destroy the protocol's security. Thus, the absence of attacks needs to be carefully considered as the verified implementation only represents a subset of an infinite space of states.

Nevertheless, the verified implementation (#System, #Actual variables) covers the majority of the specification (#Processes, #Free variables).Due to this considerable coverage a potential attack can most probably be discovered. The Casper/FDR approach is currently the best available technology and thus, the most promising method for security protocol analyses.

However, the Casper/FDR approach is not capable of explicitly analyzing the secrecy of the session-key renewal due to its limited state space. This is the main drawback of this model checking approach. Normally, only very small protocols are analyzed by Casper/FDR, such as the Needham Schroeder Shared Key Protocol. The analysis of such small protocols requires less than 30 seconds whereas the IDKE protocol needs up to 38 hours to complete the task.

Cremers and Mauw [CM04] stated that Casper/FDR is incapable of handling larger protocols due to the complexity of the graph analysis. The observations made during this work showed that the maximum memory which the FDR "*state2*" subprocess is able to allocate is approximately 3.1 Gigabyte. If only a 1GB space is required for the state space, then approximately three million graph transitions can be performed before an FDR memory allocation failure occurs. This indicates that larger protocols can be analyzed when careful design and minimized supplementing is taken into consideration.

Nevertheless, the final version realizes a secure key transfer from the pAR to the nAR using a fresh tunnel key which provides forward secrecy. The protocol was verified in order to guarantee the authentication between the MN and the next AR, as well as between the nAR and the pAR. The secrecy of all keys and nonces was also validated. Therefore, the final specification of the IDKE is precise, optimized and security validated

4.4.3 Outlook

The IDKE protocol should also provide forward secrecy and key freshness for the new session-key between the nAR and the MN (K_{NEW}). This has not been explicitly verified by the analysis due to the exponential state exploration. Nevertheless, authentication has been provided between the nAR and the MN and a nonce has been transferred securely. The deduction as to how K_{NEW} can be established under these circumstances is as follows:

The nonce can be interpreted as some potential key between the nAR and the MN. Furthermore, this key can be used for establishing a secure tunnel on which the new session-key establishment can be based and thus providing secure transfer. This also makes it impossible for a man-in-the-middle-attack since both entities have already been successfully authenticated. When the tunnel is supporting DH, this enables K_{NEW} to be established between the nAR and the MN, as well as providing forward secrecy and freshness. However, even if this approach is secure, it would involve more messages. Therefore, an optional IDKE protocol step is proposed.

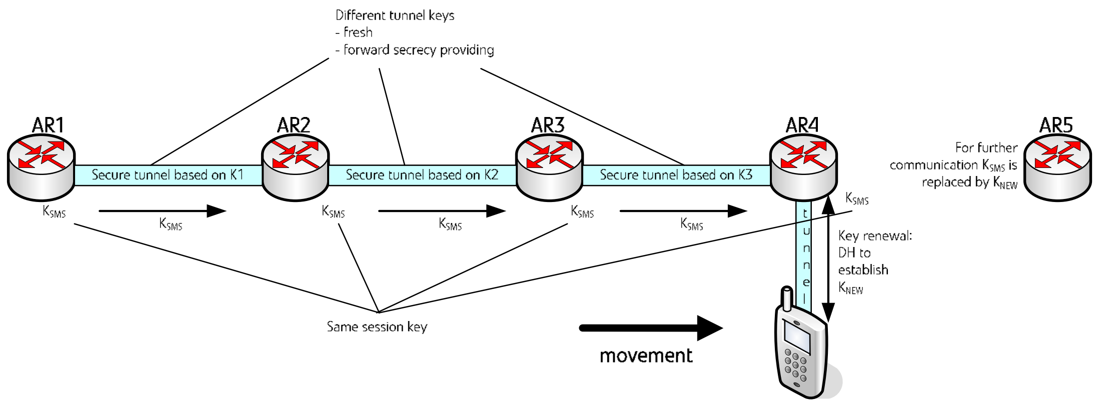

Different tunnel keys
- fresh
- forward secrecy providing
AR1
AR2
AR3
AR4
AR5
Secure tunnel based on K1
Secure tunnel based on K2
Secure tunnel based on K3
For further communication K_{SMS} is replaced by K_{NEW}
K_{SMS}
K_{SMS}
K_{SMS}
K_{SMS}
K_{SMS}
K_{SMS}
K_{SMS}
K_{SMS}
tunnel
Key renewal: DH to establish K_{NEW}
Same session key
movement

Figure 22: Session-Key Renewal

As the session-key K_{SMS} is transferred via a secure tunnel which already provides forward secrecy, this option needs only to be considered when a key renewal is desired. This might be the case when a chain of ARs has become too extended. The session-key should then be renewed as illustrated in Figure 22.

The freshness of the new session-key K_{NEW} is based on the fact that both the MN and the nAR are authenticated by agreeing on the fresh nonce *na* and the secrecy of K_{SMS}. Timestamps are not considered in the verification. It is assumed that instead of using *na* a timestamp would provide the same function, but this would cause time synchronization problems.

Chapter 5

Concurrency of IDKE Protocol Runs

In order to improve the protocol performance, this chapter deals with a protocol extension for the participating ARs. This extension aims at handling concurrent protocol runs caused by fast moving MNs. A formal description of the improvement, in particular on the AR's behavior is given by using the specification and description language (SDL). The aim of this extension is to enable an AR to deal with competing protocol runs. Thus, ARs are given the robustness of not reaching an undefined state even when multiple requests occur. This is verified by generalizing all possible cases and by simulating them with the aid of message sequence charts (MSC). This extension is compatible to the specification produced in Chapter 4. Although in this case the security properties and the robustness to counter DoS attacks have not been considered.

5.1 Introduction

Communication protocols such as the IDKE protocol involve interactions in distributed systems by interchanging messages between participating nodes. The nodes from the viewpoint of a communication protocol may act as a sender or a receiver. According to their function, these nodes can have a complex internal behavior which is described by complicated state machines, functions and local variables. Communication protocols can be examined with the aid of formal languages in order to specify, verify and validate the specified conduct. Chapter 3 introduced the IDKE protocol, while Chapter 4 showed that the entire security properties of the protocol can be fulfilled. This contemplated the participating nodes as having fixed roles (pAR as sender and nAR as receiver) by mainly focusing on a single valid protocol run. This assumed that by having just a single valid protocol run, it would also be appropriate for verifying the security properties of the protocol. However, this does not apply in the case where an MN moves so fast that a protocol run cannot be completed. Due to this circumstance, concurrent [Mil89] valid protocol runs could arise which then cause unexpected behaviors on the ARs. Thus, in the same instance, it might be considered as the nAR of protocol run *n* and as the pAR of protocol run *n+1*. Figure 23 visualizes this scenario in which the *current instance of an AR (cAR)* is considered as both a "pAR" and a "nAR". The theme of this chapter is to examine as to how an AR should behave in the event of several competing protocol runs. The major task is to provide a solution that preserves the robustness of the AR against all possible incoming messages. Robustness in this case, refers to the capability of handling even the most unexpected combinations of messages without losing the capability of performing the primary desired actions for the main protocol run. This implies that deadlocks should not occur that have an effect on any possible combination of messages for an AR. Any solution for robustness has also got to be considered from the aspects of security and performance. Therefore, this chapter concentrates on formally describing, simulating and verifying the AR's behavior with the aid of formal languages.

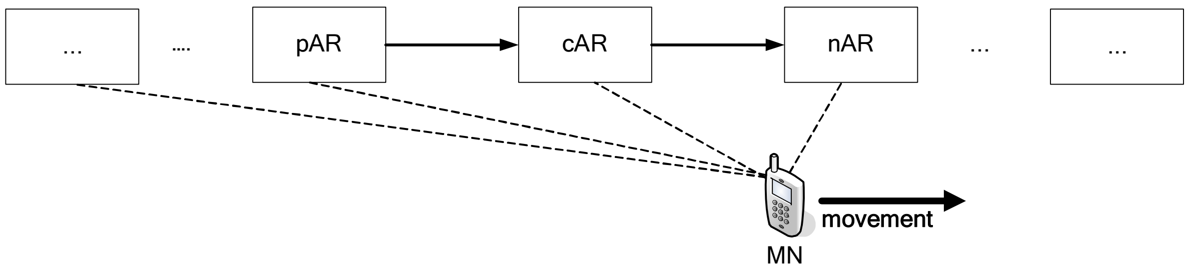


Figure 23: Fast Moving MN

5.2 Specification Languages

High-level description languages such as *Message Sequence Charts (MSC)* [ITU96] and the *Specification and Description Language (SDL)* [EHS97, ITU92, ITU99] are used to specify and to verify telecommunication systems [Kne92]. MSC and SDL have many properties in common which would suggest a possible combined use. Both mainly focus on the description of distributed systems whose components communicate by asynchronous message passing. An introduction on the main concepts of MSC and SDL is given below.

5.2.1 Message Sequence Charts

The Message Sequence Chart is a graphical specification language standardized by the ITU-T as Recommendation Z.120 [ITU96]. MSCs are used for describing the communication behavior among system components and their environment. The main concept of MSCs are *instances* and *messages*. Instances represent system components that interact with each other by exchanging messages. Instances and messages are identified by a name, while messages can optionally have parameters. Typically, each message involves two events: sending (output) and receiving (input). These events are either triggered by instances or triggered by the environment.

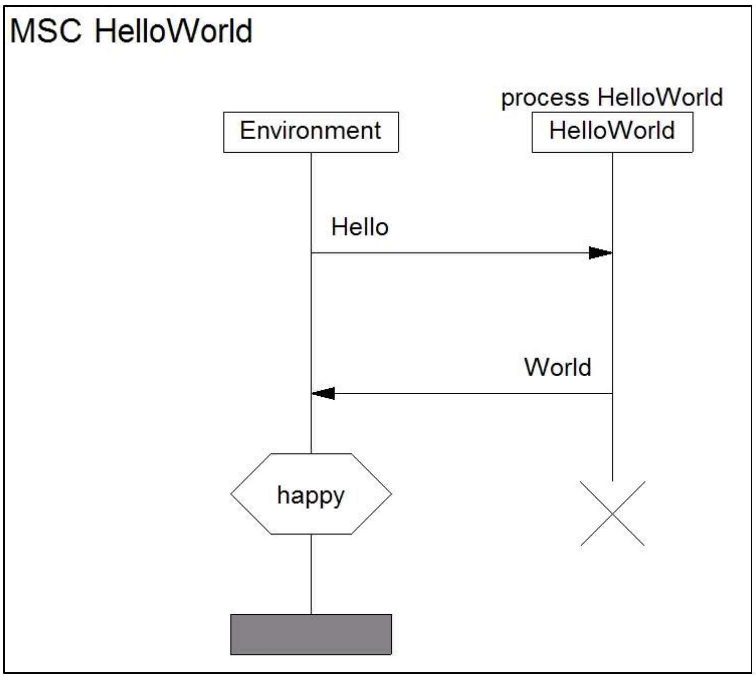


Figure 24: Hello World MSC

Figure 24 shows an example of MSCs with the instances *Environment* and *HelloWorld* displayed as vertical lines with an additional rectangle for the instance header. The horizontal bar denotes that the instance ends, whereas the cross represents the termination of the process. The MSC describes the scenario of an instance *Environment* sending a message *Hello* to the instance *HelloWorld* and receiving a message *World*. Messages are represented by annotated arrows pointing from one instance axis to another. While receiving *World*, the instance *Environment* changes its internal state to *happy*, which is expressed as a hexagon. MSCs define an absolute order along each separate instance axis. Each instance has to be interpreted from top to bottom. Any events on different instances are only partially ordered by message exchanges, which means that a message has to be sent before it can be received. Due to this, it is semantically irrelevant as to whether a message arrow points upwards or downwards even though the latter representation is more intuitive. Therefore this representation is always used within this thesis. The MSCs provide an additional function for internal actions that are represented as a rectangular box on the instance axis. These boxes may either contain a formal expression or an informal text.

5.2.2 Specification and Description Language

The *Specification and Description Language (SDL)* [ITU 92, ITU99] is a graphical language for the modeling of distributed systems. It provides the specification for the functional behavior and the structure of a system [Hog89, EHS97]. Basically, the SDL specifications can be considered as a set of *Communicating Extended Finite State Machines (CEFSMs)* which are executed in parallel, primarily performing their actions independently. The CEFSMs communicate via asynchronous messages. These are transferred via channels which connect the sender and the receiver. Channels are fundamentally unidirectional connections so that the bidirectional connections can be modeled by two channels. Each CEFSM stores variables locally. A SDL specification consists of a number of diagrams whose combination describes the hierarchical structure of a distributed system. At the top level, the SDL specifies a system which usually consists of a number of agents. These can either be blocks or processes. The two types of agents can contain further agents until the desired degree of detail is reached. Both blocks and processes include state machines, but these differ in the degree of concurrency. In blocks, the state machine of the agent is executed in parallel with its embedded agents whereas in processes the state machines are executed in an alternating manner. Transactions are interpreted atomically and sequentially.

The dynamic behavior of SDL is described by *Exceeded Finite State Machines (EFSMs)* and depends on the received messages from other EFSMs. Any incoming signals are stored in a *First In First Out (FIFO) queue.* When two signals arrive concurrently, they are stored in an arbitrary order. The queued signals are processed separately while each causes a state transition. Various actions are executed during a transition which change the internal variables. The signals are sent out throughout a transition until the next state is finally reached. Figure 25 shows a component of a SDL state machine. When in the *idle* state, an incoming signal called *hoReq* would involve a transition. Furthermore, it requires a task that sets a local variable and which sends out a *TOKENreq* signal. Finally, the transition evolves into a new state of *waitTokPar.*

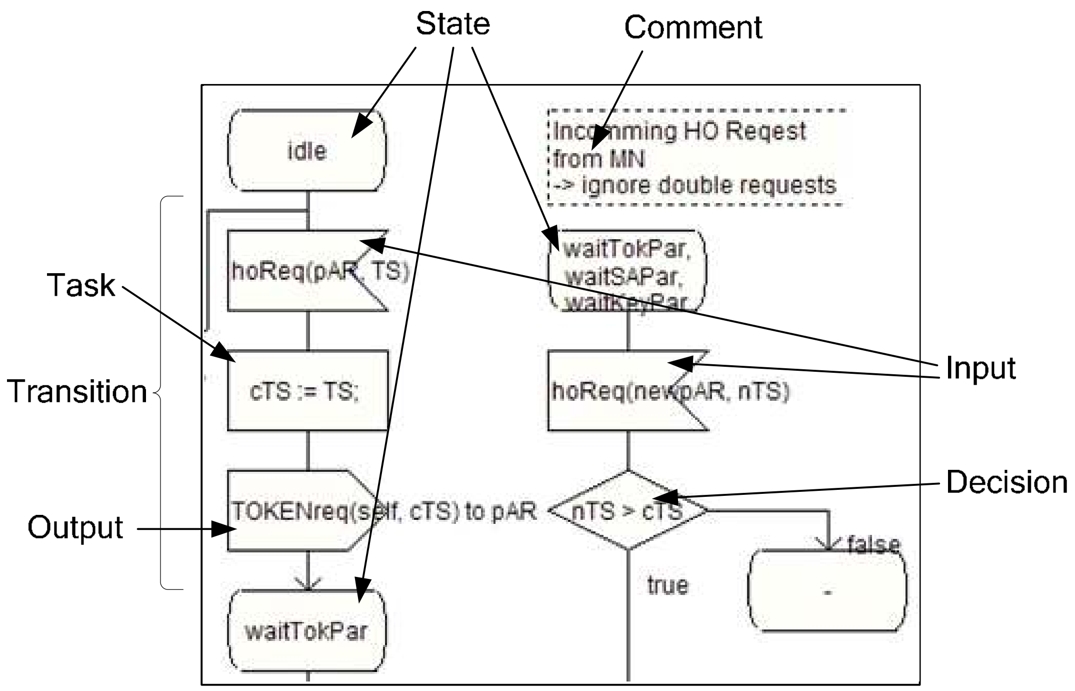


Figure 25: SDL Example

Many extensions have been made to improve the capability of SDL. The FIFO concept for queuing as an example, is interrupted when the state cannot handle the next signal as input. In this case, the inapplicable signal can either be dropped or stored separately and be executed at some later date. Other conceptions can extend the SDL by, for instance, adding timers or object orientation concepts. As these extensions are irrelevant for the evaluation carried out in this thesis they are not described further. Any reader requiring more detail can refer to [ITU99].

5.3 IDKE Node Specification

The IDKE protocol involves two ARs: one as sender and another as receiver. However, an IDKE-aware AR might adopt both roles consecutively when there is a traversing MN and thus, a former receiver becomes a future sender. It is an actuality that each IDKE-aware AR must be able to act both parts. However, most of the challenges that ARs face are in situations where several requests occur in parallel. The following scenarios are considered possible:

1. An AR receives a request from an "nAR" before having retrieved the key. In this case, an nAR is considered as a "pAR" in any further protocol run. This scenario occurs whenever an MN moves faster than the completion of a protocol run and is thus referred to as *fast moving*.

2. An MN returns to the previous AR before completing the protocol run; this is called *fast toggling*.
3. An MN double-requests the key at the same time as an AR. Although this could be a double request, the MN might also have been connected to a third AR and has as a result returned to its previous domain. This is referred to as *fast cycling*.

As examples in all three scenarios, one might imagine three ARs (AR1, AR2, AR3) where the MN is initially attached to the AR1 and performs a sequence of handovers. These handovers are assumed to take place in such a fast manner that the first one is still pending at the time the last one has been initiated. Taking the viewpoint of AR2, example sequences for all three scenarios are: "AR1, AR2, AR3" as sequence for a *fast moving* MN; "AR1, AR2, AR1" for a *toggling* MN as well as "AR1, AR2, AR3, AR1 AR2" for a *cycling* MN. Given the assumption that messages contain timestamps, the current session-key holder upon receiving several requests is always able to judge which request is the most recent one. Accordingly, the AR exclusively accepts the most recent one and cancels all others.

A difficult situation occurs whenever an AR receives a request without having the corresponding session-key. In this case, the AR is not even capable of judging as to whether the request is valid or not. It is even more challenging when an AR is waiting for a key, but instead receives a further request. The AR has three possibilities to deal with this situation:

- Drop the new request
- Queue the request
- Forward the request to the AR from which it is itself expecting to obtain the key.

The first possibility is inadequate since it would end in a key forwarding chain and would involve a second request by the MN. The second possibility is impracticable for two reasons:

- The queuing of potentially invalid requests from malicious senders would make the ARs vulnerable against DoS-attacks [Gop01].
- Treating all requests in the order of their arrival would lower the performance. The cycling scenario would specifically involve a cycling session-key transfer and would thus involve an additional delay.

Therefore, the forwarding of requests is the only possibility that corresponds with the requirements of the IDKE protocol. The design decision is accordingly made to realize the request-forwarding function as an optional part of the IDKE-AR extension.

5.3.1 IDKE AR Specification

Chapter 3 introduces and specifies all messages relating to the IDKE protocol. The major part of the protocol involves two ARs where the key is transferred between them. The pAR acts as the sender while the nAR performs the part of the receiver. According to the role an AR adopts, it performs dedicated actions within the protocol run. The tasks of the nAR are:

- to forward the MN-token,
- to establish a key for tunneling between the ARs (also called SA),
- to request the MN's session-key and
- to send an acknowledgment to the MN.

The pAR's tasks are:

- to check the MN's token in order to guarantee the MN's identity and the freshness of the request,
- to respond to the tunnel key establishment (SA request) and
- to forward the session-key.

The capability for request-forwarding and a mechanism for cancelling key transfers have to be established at each AR in addition to providing the functions for both the pAR and the nAR.

An example scenario involving request-forwarding is given in Figure 26. The scenario shows an MN fast moving among an AR-chain from AR1 to AR4. Initially, the MN was attached to AR1 and sequentially has sent handover requests to AR2, AR3 and AR4. Figure 26 shows that the key has been successfully transferred from AR1 to AR2. The tunnel establishment between AR2 and AR3 is in progress when a key-request from AR4 reaches AR3. As AR3 is unable to judge whether the request is valid or not, it forwards the request (also called token) to AR2. AR2 is in possession of the session-key and is thus able to interpret the received token as a valid request. Thus, AR2 has two tasks: Firstly, to cancel the tunnel establishment between AR2 and AR3; and secondly, to start the tunnel establishment procedure with AR4. The session-key can then be directly transferred from AR2 to AR4.

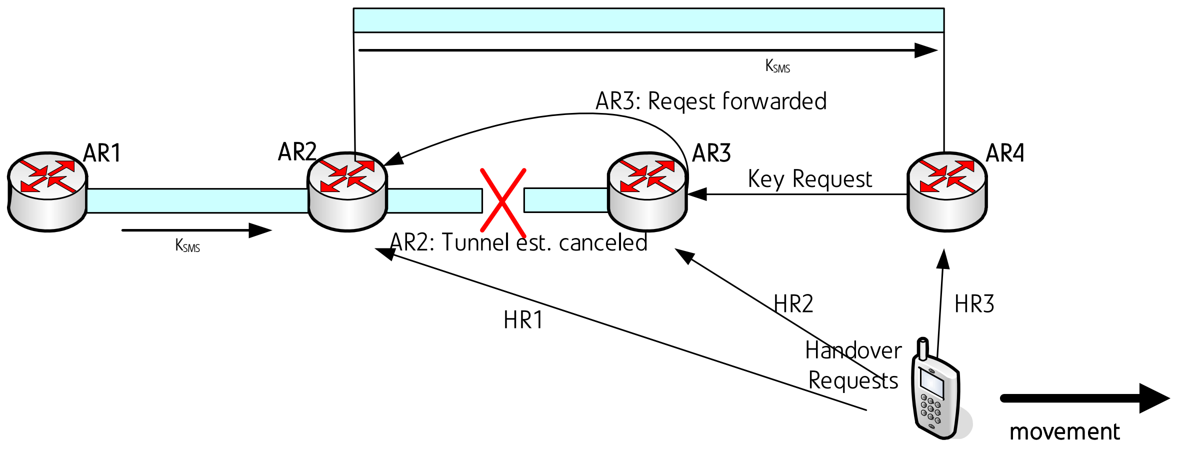


Figure 26: Example - Fast Moving MN

5.3.2 IDKE AR Extension

The fundamental concept of the extended AR specification is to initially consider both roles of sender and receiver as separate state machines. The next step is to combine them into a single state machine by first connecting the end states of one state machine with the initial state of the other. Secondly, for all states all possible incoming messages have to be checked in order to verify if they create a shortcut or not. The result is an extended IDKE-AR that is able to handle all potential IDKE messages as the sender and the receiver. It forwards `request` messages from other ARs and interrupts protocol runs by sending `cancel` messages. It is also able to accept `cancel` messages from other ARs. Figure 27 illustrates a schema of an extended IDKE-AR.

The block in the centre represents an IDKE-AR consisting of two parts: The part of the receiver and that of the sender. Initially, the AR is in the state which is referred to as `idle`. In this state, the AR has neither key nor SA. The `idle` state belongs to the receiving part of the AR. Once the MN sends an HO-Request, the AR starts the key obtaining procedure with the pAR according to the request made by the MN. After the key transfer has been performed successfully, the AR switches its state to `KEY established`. If the AR receives a token before it has obtained the key, it will forward the request to its pAR. When the token has already been forwarded, the AR sends a `cancel` message to the token forwarder. The AR likewise continues the communication and key obtaining procedure in case the token is invalid. The pAR either proves the correctness of the token and forwards a `cancel` message or it forwards the token in the same way and answers with a `cancel` message. This message resets the AR to the `idle` state.

The retrieving of the key enables the AR to enter the state of `KEY established.` This is the initial state of the sender. An acknowledgement is sent to the MN when the AR enters the state `KEY established`. The AR in this state is thus able to decide if a received token is valid and fresh. It should be mentioned here that until it receives the session-key, the AR is unable to make such a judgment. The AR now takes on the role of the "pAR" and waits for a valid key request. When such a request is received, it starts the SA-establishment and the key-transfer procedure with the nAR-ID contained in the message. One should note that the nAR-ID is not necessarily the sender of the request message since the sender could have simply forwarded the request message. In such a case, the AR sends a `cancel` message to the sender in order to interrupt the transfer.

The design choice introduced above has been selected in order to allow the internal behavior of the MN to be as simple as possible. It simply sends a request message to the current AR while the entire responsibility for the directing of messages is located at the ARs. Hence, the MN's internal behavior is very straightforward and does not require any queuing of history on the connected nodes. It merely stores the ID of the previous AR it sent a `ho request` for requesting a handover.

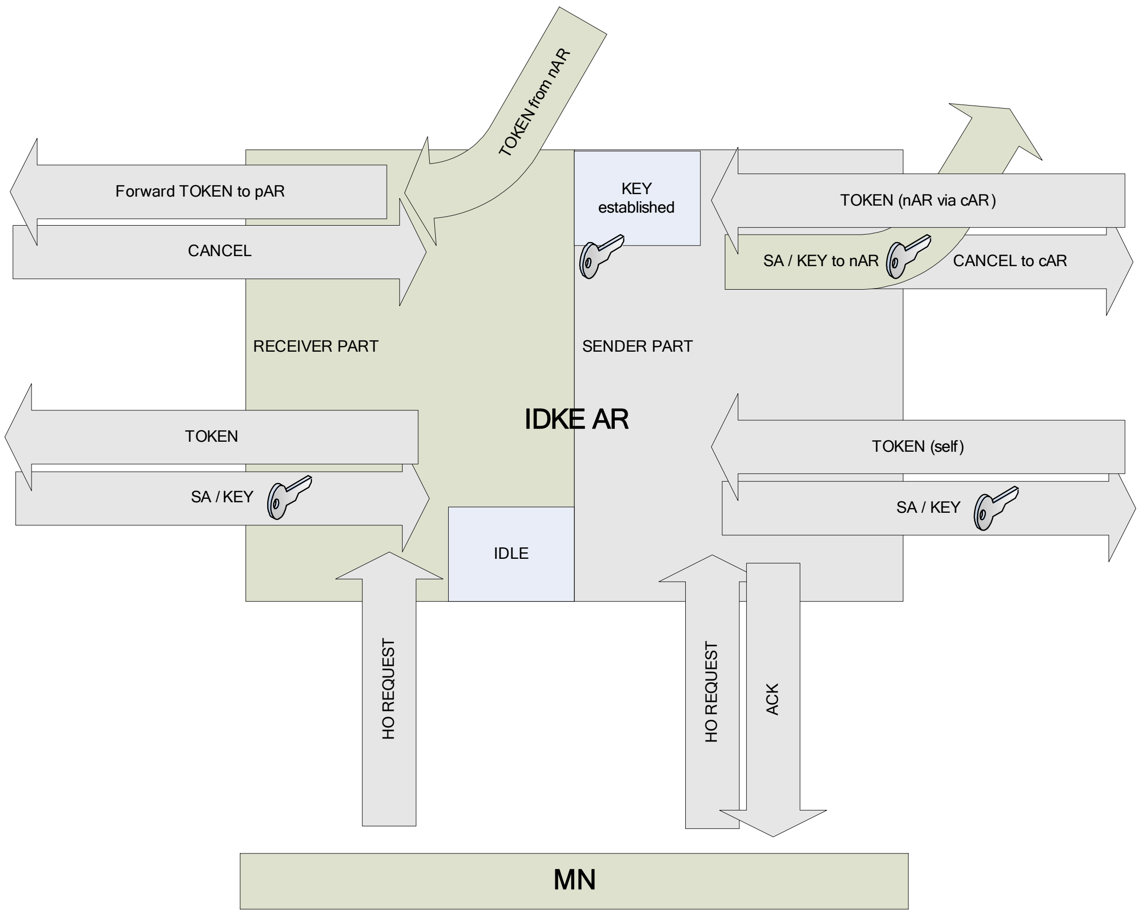


Figure 27: IDKE aware AR Overview

5.4 SDL Specification

A formal specification of the IDKE-AR as illustrated in Figure 27 utilizing the formal language SDL. The IDKE-AR consists of both a sender and a receiver which are integrated into a SDL block-type. The function of both and their interoperation are subsequently explained in more detail. A system acting as a simulation environment has also been introduced which consists of three IDKE-AR block types. All further illustrations are directly obtained by the simulation and verification environment *Telelogic Tau* [Tel01].

5.4.1.1 The Receiver Part

The SDL model for the receiver part is illustrated in Figure 28. Each session involves a unique state machine that initially inhabits the `idle` state of the receiver part. Each MN is generally involved in a single session and therefore the transfer of a single session-key per MN is required. An AR in the `idle` state expects handover request (`hoReq`) messages from the MNs that desire to have a session-key transferred to it. The `hoReq` message corresponds to message 2 of the IDKE protocol and contains two parameters, namely the pAR-ID and a timestamp. The pAR-ID determines which AR has previously received a `hoReq` from the MN. The second parameter of the `hoReq` message is the MN timestamp and is denoted as being the optional *timestamp1.* This was introduced in Section 3.2 .

The SDL model shows that the current timestamp is stored at a variable *cTS*. This timestamp is also contained in the outgoing `TOKENreq` message (message 3 of the IDKE protocol). The AR waits for the pAR to verify the token and expects either a `TOKENok` (IDKE protocol message 4) or a `cancel` message. The former indicates that the token is valid and this allows for a secure tunnel establishment. The AR sends an `SAreq` message which corresponds to message 5 of the IDKE protocol and it accordingly expects to receive message 6. This is referred to as `SAest`. Finally, the key is requested and transferred according to the IDKE protocol messages 7 and 8, herein dedicated as `KEYreq` and `KEYdata`. At this point in the procedure, the communication with the pAR is teminated and the MN is notified of a successful key transfer by the `HOack message` (IDKE message 9).

This key establishment procedure includes the following intermediated states: `waitTokPar`, `waitSAPar` and `waitKeyPar`. A `cancel` message is able to reset the key establishment procedure during all of these states. This occurs when a `cancel` message arrives which has a newer timestamp than the timestamp of the current session-key establishment procedure.

The arrival of further `hoReq` messages from other ARs also needs to be considered. As the key is not established prior to reaching the state `keyset`, the request is forwarded in all intermediate states (`waitTokPar`, `waitSAPar` and `waitKeyPar`). As can be seen from the SDL model, the key establishment procedure continues even if a newer request is forwarded. `Cancel` messages are the only means of explicitly terminating this procedure. Simulations brought out that otherwise deadlocks are possible.

While establishing a session-key, it may also be possible for the MN to again send out a `hoReq` message. When the desired pAR remains identical then only the timestamp is updated. This is in order to prevent from expired incoming `TOKENreq` or `cancel` messages from being executed. These are old messages that might have been sent out by other ARs. `TOKENreq` messages arrive whenever another AR considers this other AR as being the pAR. However, this other AR does not necessarily have possession of the session-key and consequently forwards the request. In this case, the AR modifies the destination-field that indicates where the token is to be directed. This is credible since the security verifications in Section 4.3.1 showed that the pAR-ID is not an essential part of the token. This fact allows for the receiver-ID to be placed in the non-encrypted part of the message. Therefore, this ID can be changed in transit should it be necessary to forward the request. It can be seen from the packet format (see Appendix B) that the timestamp is twice included in the message: once in plaintext and once encrypted within the authentication-token. The former timestamp acts as an identifier for determining the order in which the `hoReq` messages have arrived. Thus, the plaintext timestamp provides an order over any incoming messages even if the AR is unable to decrypt the token content. An AR can decide in accordance with this order as to which request is the most recent one. Hence, it is capable of performing the appropriate actions in exclusively dealing with the most recent one. However, the purpose of the encrypted timestamp is to enable the pAR to judge as to whether the claimed timestamp is actually valid. This is done by comparing both timestamps in order to ensure that they are equal. The timestamp in the SDL model is also represented by a session number that is simply incremented by the MN before sending a request. This simplification is practical since the purpose of the timestamp is limited to the re-sequencing of multiple requests.

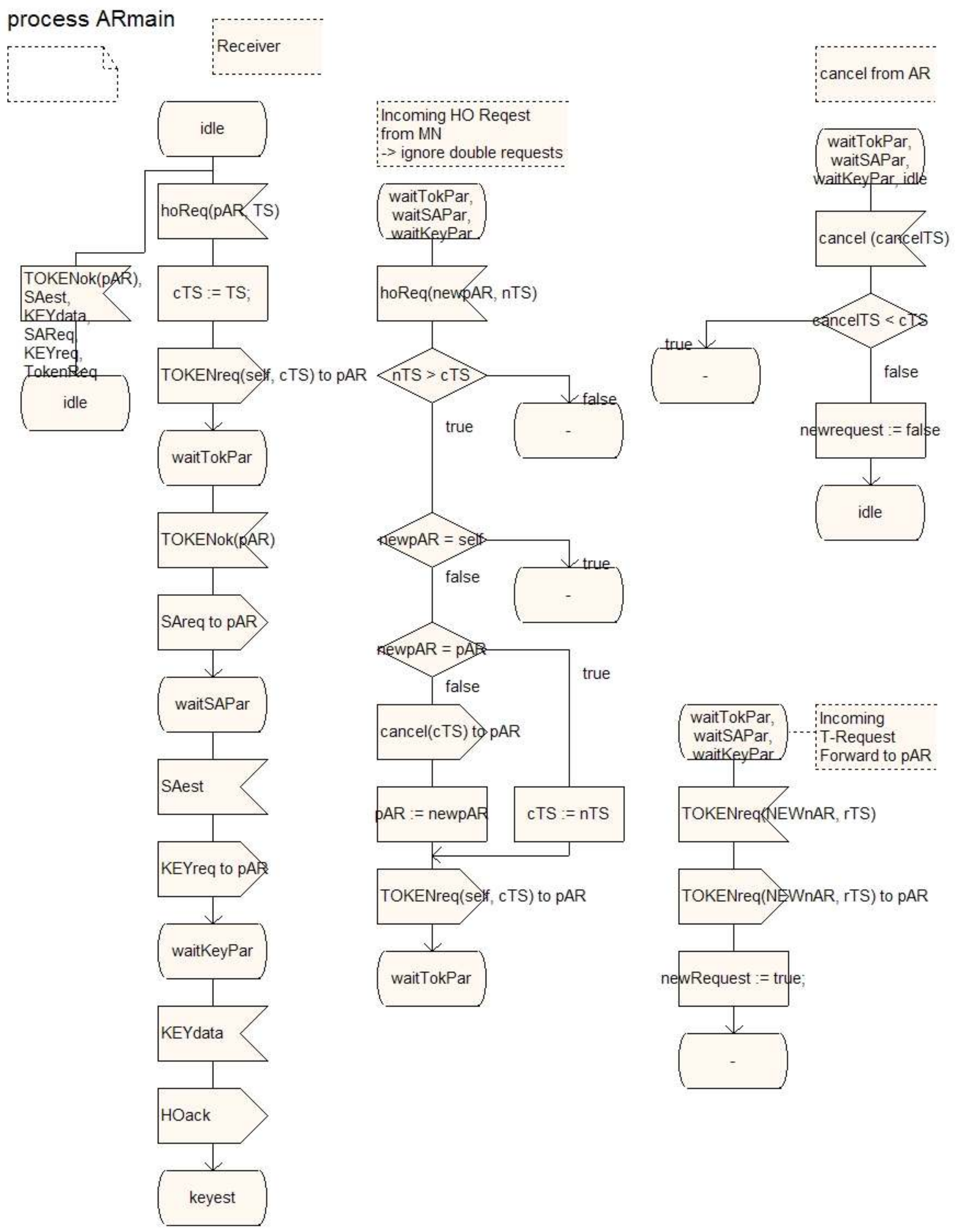


Figure 28: SDL Process Model of the IDKE-AR Receiver Part

As previously mentioned, the AR should forward all token requests to the pAR since it cannot decide as to whether the timestamp and token are valid or not. An exception is made here when the last request is stored in parallel prior to forwarding it. Simulations have been carried out that discovered cases in which the key has been sent from the pAR to the cAR parallel to the token being forwarded from the cAR to the pAR. In order to prevent the loss of this request, the cAR stores and forwards the request. Immediately after the retrieval of the key, the cAR will start the SA establishment with the nAR, if the token is valid. This trigger in the SDL model is initiated by the Boolean variable `newRequest` as illustrated in Figure 29.

5.4.1.2 Sender Part

Figure 29 shows the sender part of the AR. The `keyest` state is the initial state when the key has been successfully established and consequently the AR takes over the role of *sender*. `TOKENreq` messages are no longer forwarded, but are directly responded to. The token verification has not been modeled since any attacks are not within the scope of this particular model. Whenever an AR possesses the session-key and verifies any incoming authentication tokens, they are assumed to be valid. This means that only the freshness of the timestamp actually has to be checked.

The sender part consists of the corresponding states within the key forwarding procedure (`keyest`, `waitSA`, `waitKeyReq`). It switches back to the idle state after the key has been transferred. A new `TOKENreq` can be received during this sequential procedure run. When the request carries a newer timestamp, the forwarding procedure is interrupted and a `cancel` message is sent to the sender. In such cases, the sequential process is re-started for the new request and a `tokenOK` message is sent out.

All messages are queued in this SDL model. A message is not dropped in case of a parallel arrival. This is the reason why the `SAreq` and the `KEYreq` messages are explicitly dropped when the sender is in `keyest`. It is illustrated in Figure 29. The `hoReq` messages sent by the MN are immediately answered with a `hoAck` message in case an MN has send out a double request.

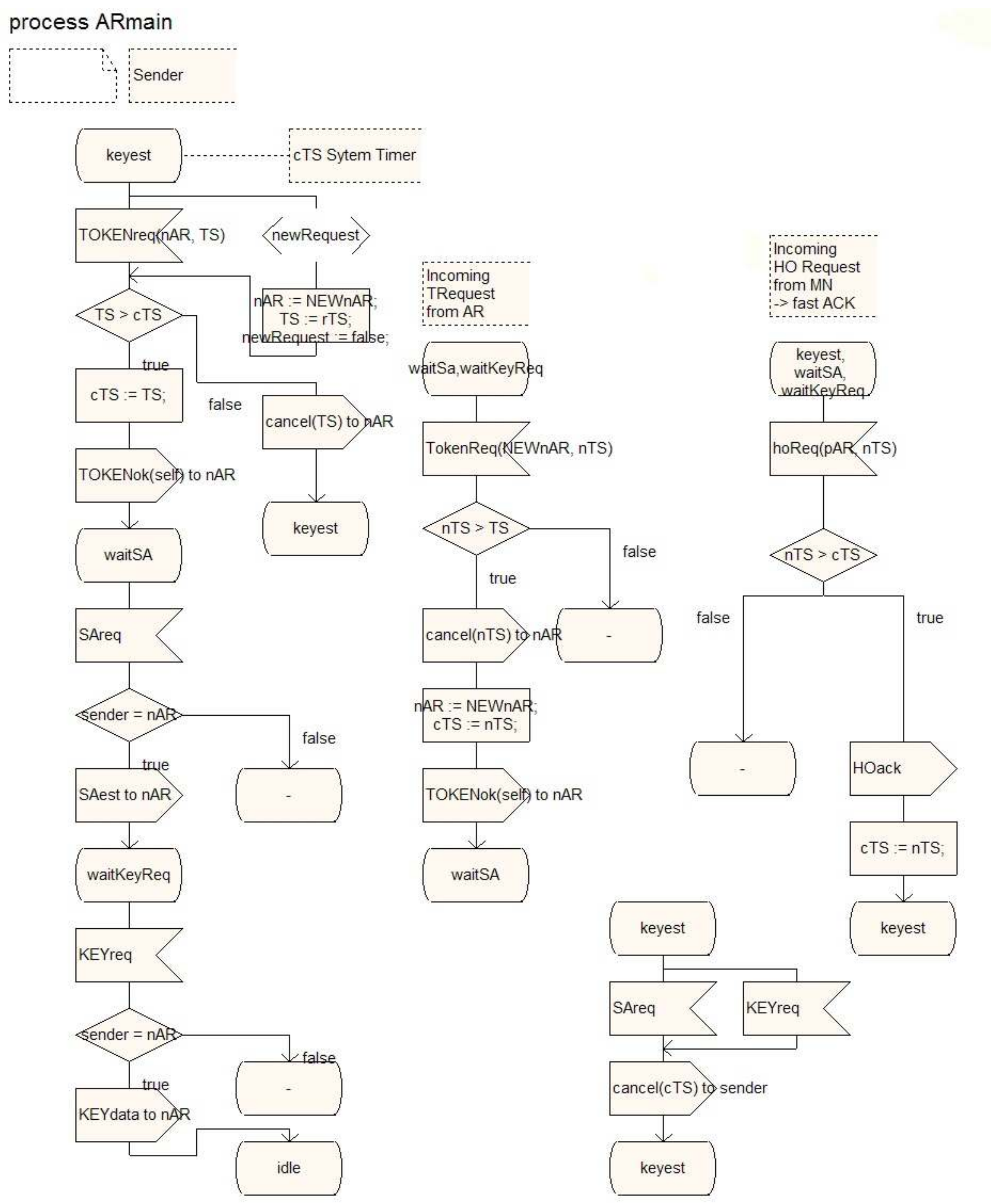


Figure 29: SDL Process Model of the IDKE-AR Sender Part

5.4.1.3 Mobile Node

The MN specification is illustrated in Figure 30. As one can see, it has been kept as simple as possible. Subsequent to the initialization procedure, the MN sends out the `hoReq` messages by means of an external trigger. The external trigger is used for testing purposes and is initiated during the simulation process. In this particular SDL model, the trigger is not specified in more detail as the handover decision has not been included in the IDKE protocol. Hence, a `hoReq` message is sent to the nAR. This message contains two variables, namely the pAR-ID and the timestamp. It was decided in its design to handle all the forwarding logic at the ARs and the MN is simply used for storing the last AR it has sent a `hoReq` to. The timestamp is implemented as a session ID counter which is incremented prior to performing the next handover. The MN also receives `hoAck` messages and stores the sender as the current AR. It should be mentioned here, that the current AR-ID is needed for future purposes. Due to an external trigger, it is also possible that the MN wants to handover to the AR it is currently connected to. Such cases need to be handled by the logic implemented at the ARs.

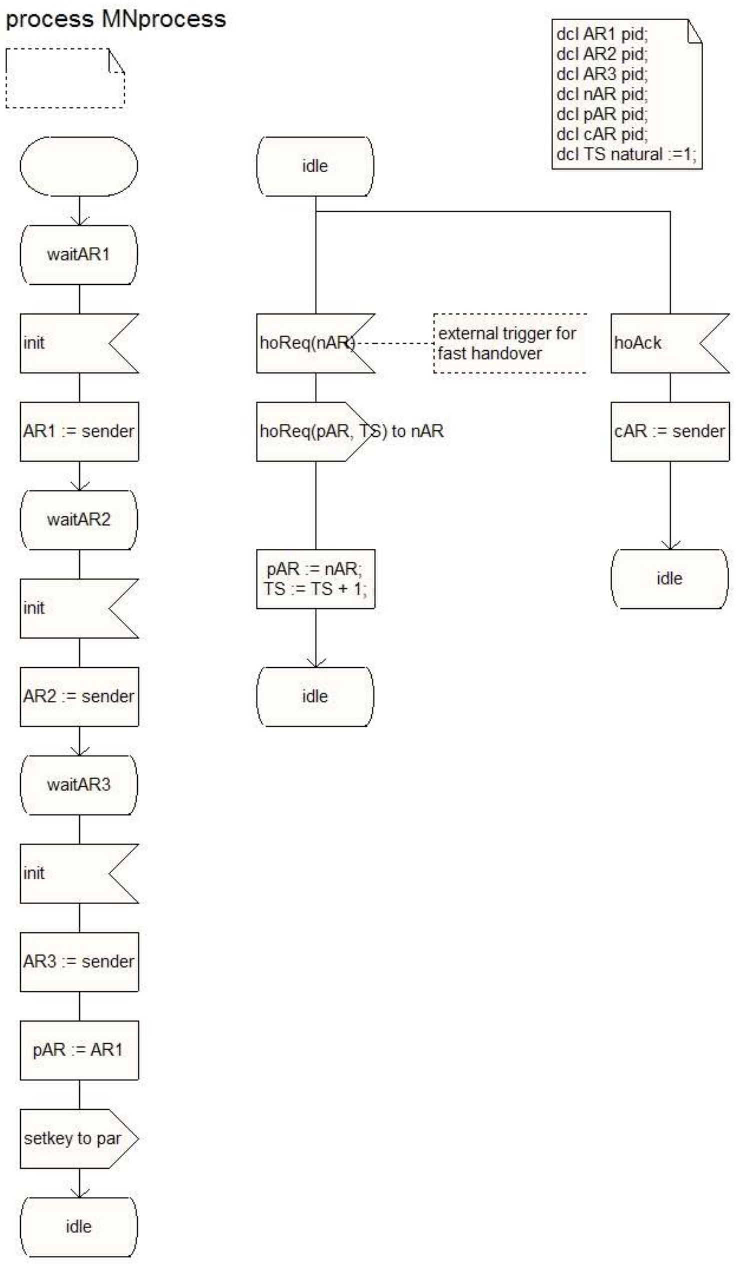


Figure 30: SDL Process Model of the IDKE-MN

5.4.2 Scenario-based Simulation

The IDKE system is illustrated in Figure 31. The figure illustrates the IDKE-AR modeled within an SDL block type. Furthermore, it shows the three instances of this block type as AR1, AR2 and AR3. The creation of new scenarios is simplified, as the block type can be reused for any new instance of an IDKE-AR. In utilizing the block type thus, it improves the scalability for creating more complex scenarios.

All ARs are connected with each other. The signals contained within the `ARsignals` list can be exchanged, as the example scenario shows in Figure 31. The MN merely sends `hoReq` messages to the AR and expects a `hoAck` back from the AR. Initially, the key is set at AR1 by sending a `setKey` from the MN. This is applied for testing purposes.

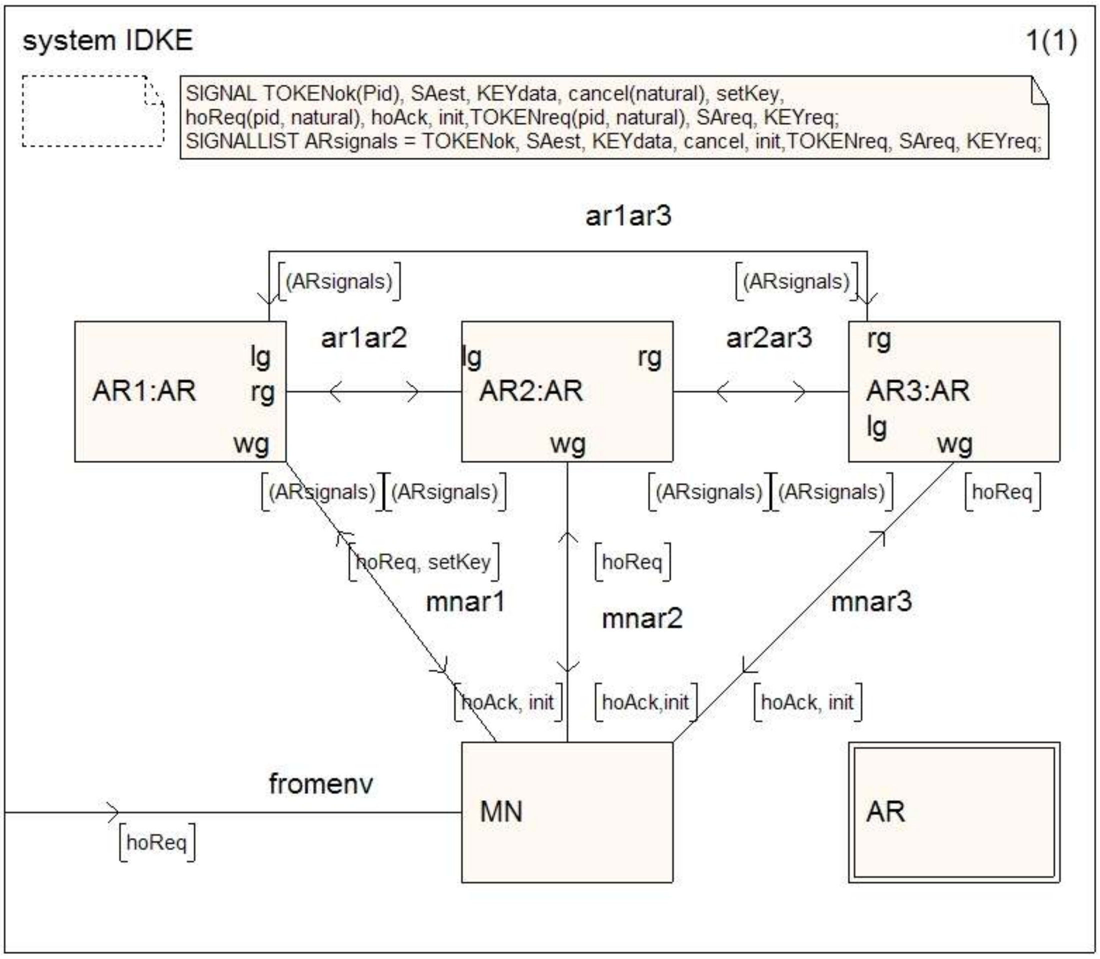


Figure 31: IDKE SDL System

This simple scenario demonstrates the possibility of creating a considerable number of different uses cases. Apart from the simple sequential handover, the following special cases have been examined:

- ***Fast movement:*** the MN sends very fast consecutive requests.
- ***Cycling:*** the MN moves fast between the ARs: for instance from AR1 to AR2 and AR3. It then returns back to AR1. The request either cycles behind the key or can even overtake the key.
- ***Double request:*** The MN sends several requests to one AR. As for example in the following sequence: AR1, AR2, AR2 and AR3.
- ***Toggling MN:*** In this case the MN fast switches between two ARs, as for instance in: AR1, AR2, AR1, AR2 and AR1.
- ***Combinations:*** The MN may produce several parallel requests, including toggling, fast movement and double requests.

5.4.2.1 Fast Movement

The task of the first simulation is to analyze a fast moving mobile MN. The MN performs two handovers in such a fast manner that the first has not finished, when the second handover is initiated. The objective of the simulation is to verify the specification against any deadlocks and livelocks. The MSC resulting from the simulation is presented in Figure 32. It shows an MN that was initially connected to AR1 and which performed overlapping handovers to AR2 and AR3. This MSC also shows the intermediated states of all instances. Figure 32(a) illustrates how the AR2 obtained a `TOKENreq` from AR3 and how the request is forwarded to AR1. Subsequent to forwarding the token, the AR2 continues to communicate with AR1 (see Figure 32 (b)). This is because the `SAest` message from AR1 arrives after the `TOKENreq` message from AR3. Consequently, theAR2 sends a `KEYreq` message to AR1. However, after receiving the token from AR3, the AR1 sends a `cancel` message (see Figure 32 (c)) to AR2 and so the AR2 returns to the `idle` state without deadlocking. Finally, the key is directly transferred from the AR1 to AR3 (see Figure 32 (d)) and in doing so the AR3 reaches the `keyset` state while all other ARs are in the `idle` state.

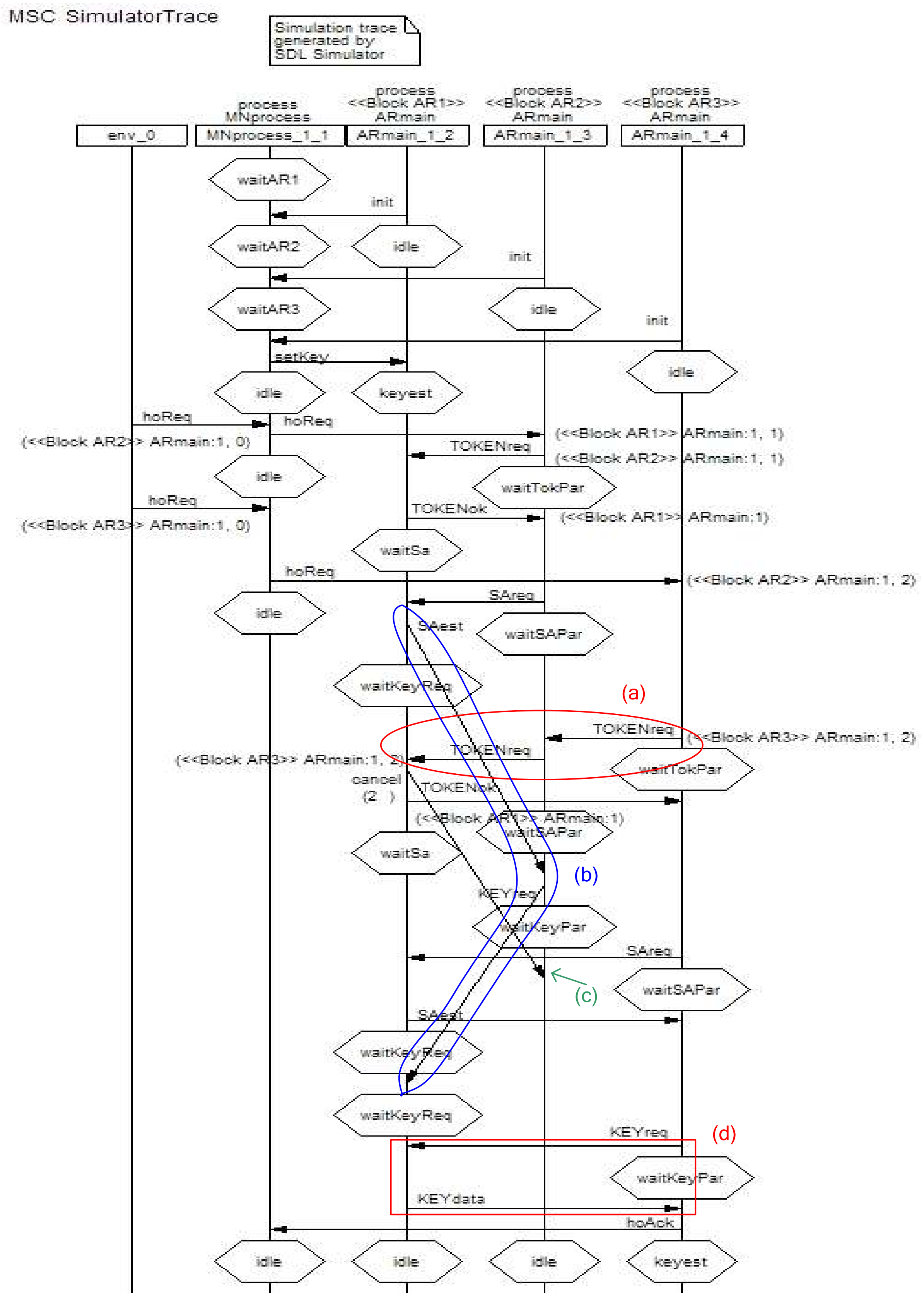

MSC SimulatorTrace
Simulation trace generated by SDL Simulator
env_0
process MNprocess
MNprocess_1_1
process <<Block AR1>> ARmain
ARmain_1_2
process <<Block AR2>> ARmain
ARmain_1_3
process <<Block AR3>> ARmain
ARmain_1_4
waitAR1
init
waitAR2
idle
waitAR3
setKey
keyest
hoReq
(<<Block AR2>> ARmain:1, 0)
(<<Block AR1>> ARmain:1, 1)
TOKENreq
(<<Block AR2>> ARmain:1, 1)
waitTokPar
TOKENok
(<<Block AR1>> ARmain:1)
(<<Block AR3>> ARmain:1, 0)
waitSa
(<<Block AR2>> ARmain:1, 2)
SAreq
SAest
waitSAPar
waitKeyReq
(a)
(<<Block AR3>> ARmain:1, 2)
cancel (2)
KEYreq
(b)
waitKeyPar
(c)
(d)
KEYdata
hoAck

Figure 32: MSC - Handover from AR1 to AR2, AR3

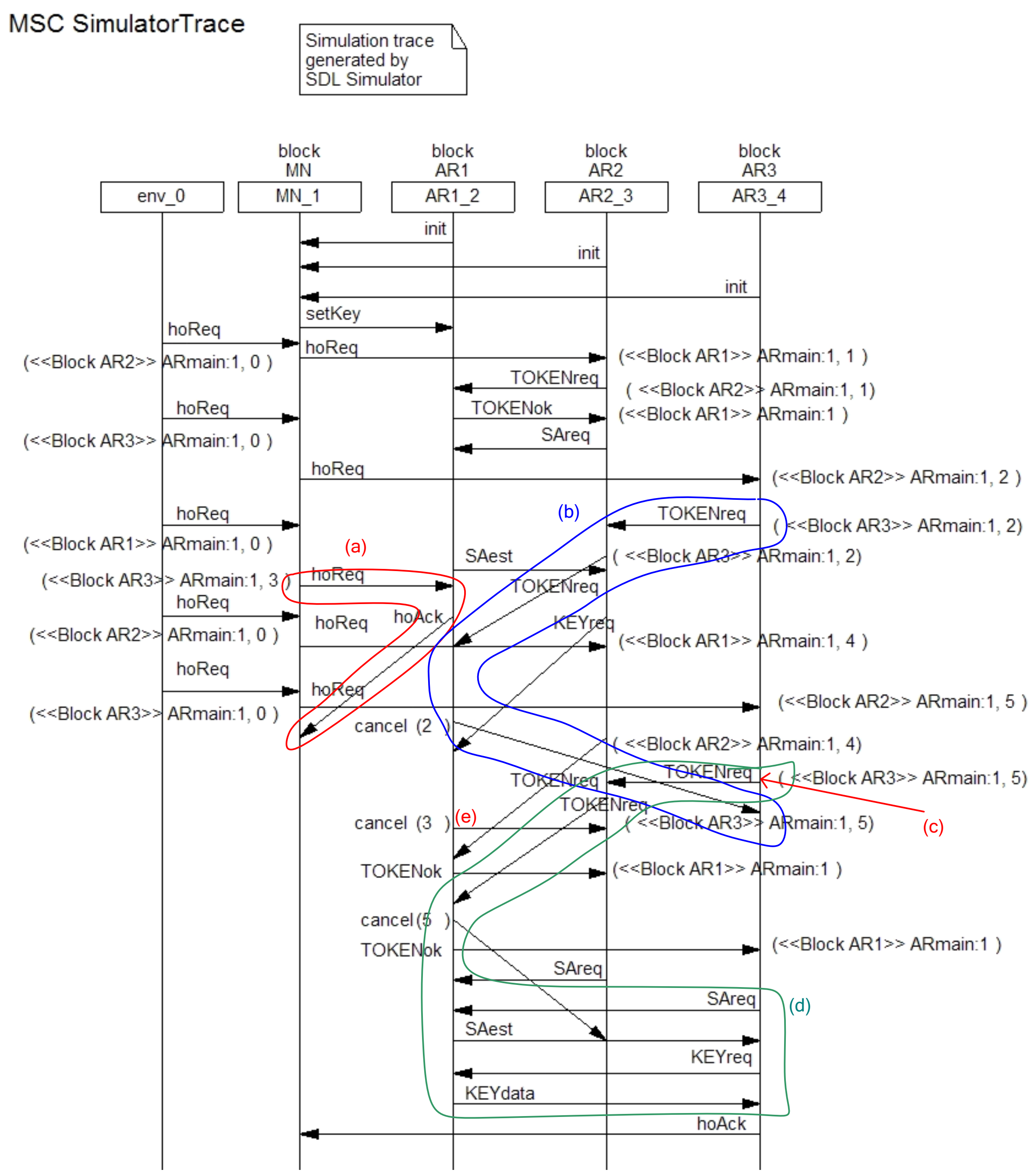

MSC SimulatorTrace
Simulation trace generated by SDL Simulator
block MN
block AR1
block AR2
block AR3
env_0
MN_1
AR1_2
AR2_3
AR3_4
init
setKey
horeq
hoReq
TOKENreq
TOKENok
SAreq
SAest
KEYreq
KEYdata
hoAck
cancel (2)
cancel (3)
cancel (5)
(<<Block AR2>> ARmain:1, 0)
(<<Block AR3>> ARmain:1, 0)
(<<Block AR1>> ARmain:1, 0)
(<<Block AR3>> ARmain:1, 3)
(<<Block AR1>> ARmain:1, 1)
(<<Block AR2>> ARmain:1, 1)
(<<Block AR1>> ARmain:1)
(<<Block AR2>> ARmain:1, 2)
(<<Block AR3>> ARmain:1, 2)
(<<Block AR1>> ARmain:1, 4)
(<<Block AR2>> ARmain:1, 5)
(<<Block AR2>> ARmain:1, 4)
(<<Block AR3>> ARmain:1, 5)
(a)
(b)
(c)
(d)
(e)

Figure 33: MSC - Handover from AR1 to AR2, AR3, AR1, AR2, AR3

5.4.2.2 *Cycling Movement*

A more complex handover scenario is when the cycling MN arrives at some AR for the second time round even before this AR is able to finish the first protocol run. This scenario is illustrated as block trace and for the sake of clarity, the states have been removed in Figure 33. Here, an MN performs a handover among AR1, AR2, AR3, AR1, AR2 and finally reaches AR3. This scenario shows a double loop over all three

ARs. The loop is so fast that the key has not been transferred between any AR by the time the MN has finished the loop.

When the MN finishes the first loop and sends a `hoReq` to AR1, the AR1 immediately responds with a `hoAck` as in Figure 33(a). The AR1 also receives a request from AR3 (forwarded by AR2). As the timestamp of the request is older than the current run at AR1, it sends a `cancel` message `cancel(2)` to the AR3 as presented in Figure 33(b). The number in brackets represents the timestamp and thus `cancel(x)` belongs to the protocol initiated at time `x`.

When the `cancel(2)` arrives at AR3, it has already sent out a further `TOKENreq` message as illustrated in Figure 33(c). Due to the timestamp in the `cancel` message, the AR3 ignores the `cancel` message as it belongs to a previous run. It continues the key establishment procedure with AR1 in order to finally obtain the key from AR1 as presented in Figure 33(d).

During the key request of AR2 from AR1, a `hoReq` from AR3 addressed to the AR1 arrives at AR2 and is forwarded to AR1. Nevertheless, the AR2 continues the transfer process and ignores the `cancel(3)` message from AR1 (illustrated in Figure 33(e)). This is because the `cancel` results from a previous request from AR3 and thus the AR2 continues the request with timestamp 4. When the newer cancel message `cancel(5)` arrives at the AR2, it ceases to communicate with AR1.

Due to race conditions during the transfer of `cancel(5)`, two SA requests arrive at AR1. However, an answer is only given to the second request originating from AR3. Finally, the AR3 is in the `keyest` state (not illustrated in the figure) and all other ARs are in the `idle` state.

5.4.2.3 Toggling Movement

This scenario does not involve the AR3 at all, as the MN toggles between the AR1 and the AR2. The scenario is similar to the cycling scenario and hence it can be considered as a cycle between two ARs. The difference is that from the viewpoint of an AR, all messages are exclusively exchanged with only one other AR.

In Figure 34, a toggling handover is shown in which the MN that was initially connected to the AR1 is fast toggling to AR2, AR1, AR2, AR1 and finally to AR2. As illustrated in the MSC, the second `hoReq` sent back to AR1 causes the AR1 to send a `hoAck` to the MN (see Figure 34(a)). The AR1 also sends a `cancel(2)` message to AR2 after the AR2 has sent a `KEYreq` message to AR1 (see Figure 34(b)). According to the timestamp in the cancel message, the AR2 terminates the key establishment procedure. The AR1 is successfully in the state `keyest` while all other ARs are in the idle state (not illustrated in the figure).

The MN now toggles faster so that the hoReq messages are sent with less delay than previously. The MN sends requests to AR2, AR1 and AR2 as fast as possible (see Figure 34(c)). Thus, the `cancel(3)` message relating to the first request arrives even before the `TOKENok` message of the fifth protocol run. Consequently, the AR2 ignores `cancel(3)` and successfully completes the key establishment procedure with AR1. Finally, the AR2 inhabits the `keyest` state while all other ARs are in the `idle` state.

Therefore, despite the toggling speed, the ARs are able to correctly handle handover requests and it is possible for the session-key to be sent to the AR which made the most recent request.

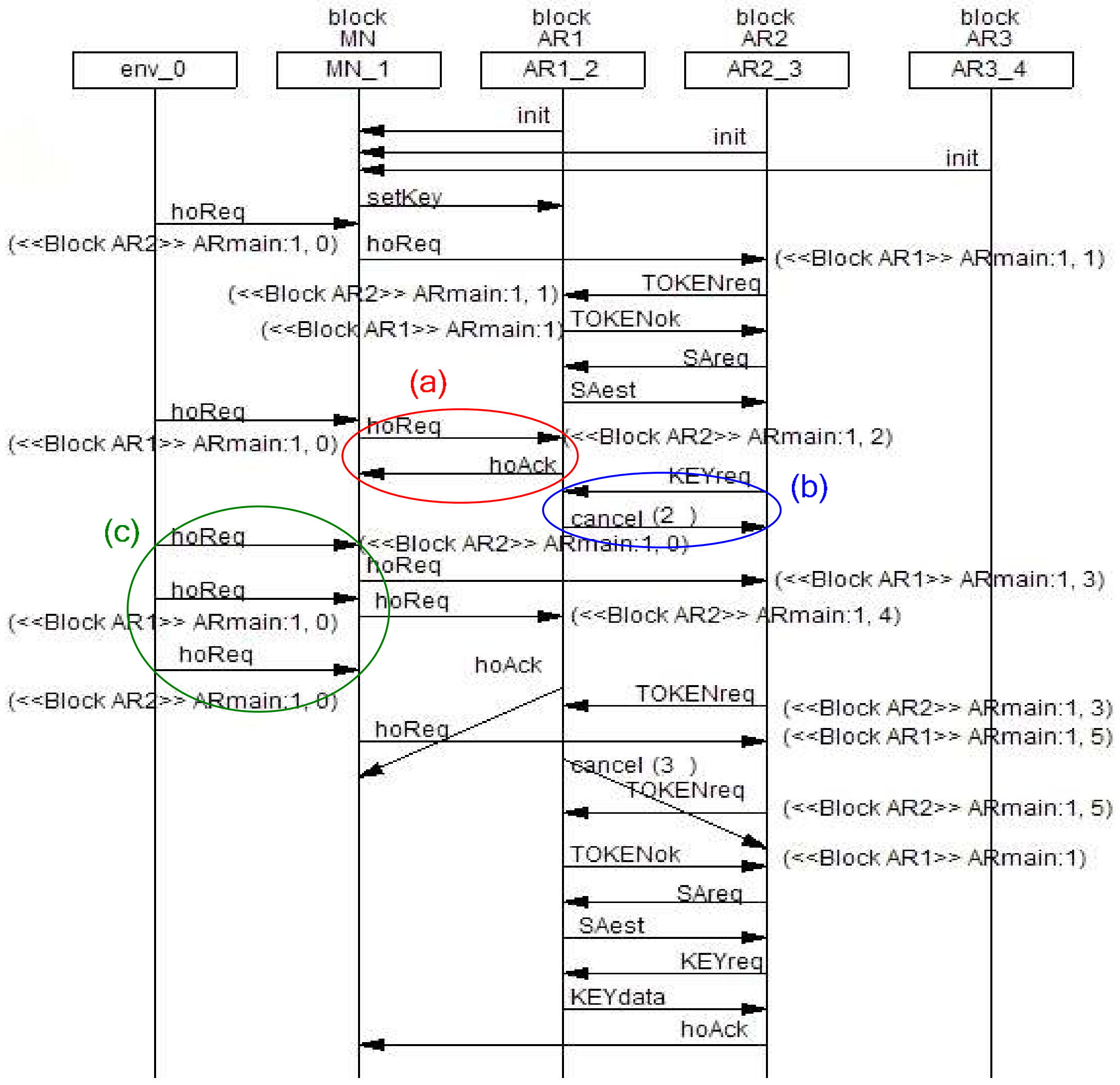

block MN
block AR1
block AR2
block AR3
env_0
MN_1
AR1_2
AR2_3
AR3_4
init
init
init
setKey
hoReq
(<<Block AR2>> ARmain:1, 0)
hoReq
(<<Block AR1>> ARmain:1, 1)
TOKENreq
(<<Block AR2>> ARmain:1, 1)
TOKENok
(<<Block AR1>> ARmain:1)
SAreq
SAest
(a)
hoReq
(<<Block AR1>> ARmain:1, 0)
hoReq
(<<Block AR2>> ARmain:1, 2)
hoAck
KEYreq
(b)
cancel (2)
(c)
hoReq
(<<Block AR2>> ARmain:1, 0)
hoReq
(<<Block AR1>> ARmain:1, 3)
hoReq
hoReq
(<<Block AR1>> ARmain:1, 0)
(<<Block AR2>> ARmain:1, 4)
hoReq
hoAck
(<<Block AR2>> ARmain:1, 0)
TOKENreq
(<<Block AR2>> ARmain:1, 3)
hoReq
(<<Block AR1>> ARmain:1, 5)
cancel (3)
TOKENreq
(<<Block AR2>> ARmain:1, 5)
TOKENok
(<<Block AR1>> ARmain:1)
SAreq
SAest
KEYreq
KEYdata
hoAck

Figure 34: MSC - Handover from AR1 to AR2, AR1, AR2, AR1, AR2

5.5 Summary, Discussion & Outlook

This chapter specified IDKE aware ARs utilizing SDL. The topology to simulate fast moving, cycling and toggling MNs consists of three ARs. By using *Telelogic Tau*, it was possible to verify that this extended specification of the IDKE protocol is robust against deadlocks and livelocks as well as against message race conditions. The AR is modeled as an SDL block type. This makes scalability an easy task if changes to the topology are required.

The IDKE protocol was extended in order to handle concurrent protocol runs. The solution of forwarding messages reduces the latency and thus provides a higher performance. The overlapping of protocol runs were distinguished by timestamps that denoted the moment when the MN had initiated the protocol run. The results ascertained by the simulation are as follows:

- Roles are separated from instances, as an IDKE-AR is represented as a single SDL block type. The analysis is even more realistic since an IDKE-AR can adopt both the roles of "pAR" and "nAR". Concurrent protocol runs require an IDKE-AR to inhabit both roles simultaneously.
- Scalability is supported as the SDL block type can simplify the creation of new scenarios. The scenario portrayal of three ARs can simply be expanded by inserting new ARs derived from the block type.
- Simulations indicated that the analyzed SDL model is robust against deadlocks. Many previous versions resulted in undesired states.
- The system consisting of three nodes was discovered to be apt in handling a vast variety of possible scenarios. The analyses proved that a fast moving, cycling, or toggling MN can be simulated.
- Cancel messages extended the IDKE protocol. These messages turned out to be necessary for cancelling previous protocol runs.
- Timestamps were shown to be required instead of nonces. Simulations showed that timestamps are mandatory for distinguishing the protocol runs. Simulations illustrated that cancel messages can arrive delayed. The carrying timestamps add the capability for the AR to decide as to whether these messages can be neglected or not. Thus, timestamps make a major contribution to providing robustness in cases of race conditions due to the overlapping of handovers.

Therefore, the extended IDKE protocol is capable of handling concurrent protocol runs. The IDKE protocol ensures that no SA is established and no key is transferred when the nAR holds an invalid token.

One major drawback to the simulation is that the token verification is not part of the SDL model. It is assumed that the token verification is performed at the sender side of the AR's model in parallel for verifying the timestamp. The protocol is robust against DoS-attacks that attempt to establish unnecessary SAs between the ARs and against any unnecessary states at the AR. Thus, ARs are robust against DoS attacks. However, a disadvantage is that any DoS attacks against the MN have not been considered in this study. Protection does not exist for the prevention of unauthorized protocol termination. A concept of authenticating cancel messages does not exist at the present time.

Subsequent work could further improve the SDL model, such as:

- Including the key computation function in the SDL model.
- Adding the possibility of reusing SAs in the SDL model.
- Including the home network in the model. Thus, providing an additional key establishment from a fully trusted instance.
- Including a retransmission mechanism for enabling the protocol to be robust against package loss.
- Utilizing the *UML security (UMLSec)* [Jue04] approach which is also based on CSP and FDR model checking.

Chapter 6

Performance Simulations with OPNET

This chapter analyzes the performance of the IDKE protocol by comparing it to other protocols. The aim is to evaluate the conditions under which the protocol provides the best results. Therefore, the IDKE protocol has been compared with a traditional challenge-response mechanism such as in GSM. The performance of an EAP and AAA-architecture based approached has also been considered. Here, the wireless shared key exchange protocol (W-SKE) is a key exchange protocol which is especially designed for wireless nodes, is also used for comparison with the IDKE protocol. The evaluation focuses on the performance of the protocols and in particular on the overall handover delay. These analyses act as a proof of concept in order to stress the circumstances under which the IDKE protocol is the most promising approach. Therefore, these studies have been separated from security-analyses and scalability-aspects. Thus, they do not consider any authentication-token computation, encryption or decryption, nor any prevention from Denial- of-Service-attacks.

6.1 Introduction

Mobile participants desire seamless connectivity while moving. Handovers become necessary and cause connection reestablishments. Delays are caused due to reauthentication procedures. This process is particularly time-consuming when located far from the home network in terms of hops so that which causes a long delay for transferred traffic. Thus, locally performed authentication runs are expected to provide better performance. The IDKE protocol on the whole exchanges messages for reauthentication locally. Other protocols that are widely used and highly involve the home network are for example, the GSM [Hei98] and the W-SKE [SBG+03]. Intuitively, the IDKE protocol should be faster when utilized far from the home network.

However, simulations are required to predict the conditions under which the IDKE protocol is faster than other approaches. Thus, the discrete event simulator OPNET Modeler [Opn05] has been utilized for this purpose. The task is to calculate the performance of the protocol in relation to other approaches. The leading question is in which environments the IDKE protocol can provide better quality than others. Quality refers to a short overall handover delay.

An inter-domain handover has been modeled as a reference scenario. Protocols have been simplified and all performance irrelevant parts have been neglected. The purpose of the analyses is to work out the following:

- The expected overall handover latency. Analyses are dependent on the performance of intermediate links. Considered links are: a wireless connection, a core network, an access network and a link to the home network.
- The performance of other widely used protocols. The analyses study the GSM challenge response mechanism and the EAP based wireless shared key exchange protocol (W-SKE).
- The simulation results by comparing all of the tree approaches. The purpose is to identify the conditions under which each protocol can provide the best quality.

The simulation tool OPNET Modeler provides both, already existing models and the capability of implementing one's own protocols. The existing models provide a set of links and nodes which makes it convenient for creating a topology. However, when one produces one's own implementations this requires a finite state machine description, a link and packet format definition as well as c-code implementation. Therefore, producing one's own implementations is the only practical selection for carrying out the performance evaluation of the IDKE protocol. All protocols approaches have been reengineered for a number of reasons:

- Models do not exist for the IDKE protocol.
- The same infrastructure has to be used in order to compare all protocols. Thus, the same nodes and links are used for all protocols.
- The function should be kept as simple as possible.

All irrelevant aspects have been removed for the performance simulation. An encryption or decryption for example, causes a delay which is modeled within the nodes, because a real cryptographic algorithm is not implemented at the node. The messages are sent unencrypted since the actual content does not have any influence on the transit-performance. All session identifiers, nonces and keys are actually not created for the packets. Random numbers are utilized to fill packets. Hence, the protocol implementation becomes less complex, but at the expense of the scalability of the simulation scenario. This is a reasonable design choice since the focus is on a handover delay without any modeling package loss and retransmission capabilities. The simulation of such protocol robustness as that of resistance against denial-of-service-attacks has not been included in the evaluation and thus, it has been assumed that the nodes will provide this desired service. Therefore, the simulation environment provides an optimal basis and similar conditions for all protocols. The overall performance of the protocols only depends on changing the link quality during the simulation run.

The following four protocols have been examined in the evaluation: the GSM, the W-SKE and two versions of the IDKE protocol. While one IDKE version performs key exchange locally, the other includes the home authentication procedure.

Further in this chapter, the whole range of mechanisms is introduced prior to describing the topology. It explains the changes made to the protocols in order to adapt them to the simulation environment. Some additional details are also given on topology as well as presenting information on the simulation runs. Finally, the simulation results are summarized and comparisons are made on the weaknesses and strengths of all of the protocols.

6.2 Related Work

The evolution of mobile phones has also had an influence on security properties as well as their strengths. This applies to the first versions on the market right up to the *fourth generation (4G)*[Bri01]. The *first generation (1G)* were analog cell phones without any encryption. GSM as an instance of the *second generation (2G),* was originally designed with a moderate level of security. Security tasks are user authentication by utilizing shared-secret cryptography, which is also used for encrypting the communications between the mobile terminal and the base station. Furthermore, the deployment of the *third generation mobile phones (3G)* [Kor01, PMK00], adopted the *Universal Mobile Telecommunications System (UMTS)* [Mah98, BA02] and introduced an optional *Universal Subscriber Identity Module (USIM.)* This uses a longer authentication key for improving security and mutually authenticating both the network and the user, whereas GSM only authenticates users to the network. However, although the security model offers data confidentiality and authentication, it is limited in its authorization capabilities and does not offer any non-repudiation.

GSM incorporates a suite of cryptographic algorithms in order to provide security. Algorithms referred to as A5/1 and A5/2 [XHW94] are stream ciphers that are employed for ensuring over-the-air voice privacy. The A5/1 is a stronger algorithm used within Europe, whereas the A5/2 is a weaker version used in other countries. However, serious weaknesses have been found in both algorithms. It is possible to break the A5/2 in real-time, even when only the cyphertext is known (cyphertext only attack). However, GSM is a very simple approach that is supposed to provide high performance. The same is true of the W-SKE approach which aims to limit message exchanges between the access network and the home network. Thus, the GSM, the W-SKE and the IDKE protocol are suitable candidates for being used as comparisons under performance aspects. Both the GSM key establishment and the W-SKE are very different from the IDKE protocol and thus are explained in detail in the next section.

6.2.1 Global System for Mobile Communications (GSM)

The *Global System for Mobile Communication (GSM)* was developed in the second generation (2G) of mobile phones and has become a well-known example of mobile communication protocols. The GSM protocol includes an authentication and key exchange mechanism for distributing a session-key between the MN and the access network. When an inter-domain handover takes place, the GSM protocol performs a total reauthentication based on a challenge response mechanism. As the GSM protocol uses one very simple mechanism, it is assumed that it can provide high performance. Due to the small number of messages, it is supposed to quickly establish a session-key. However, the GSM protocol does not provide mutual authentication. Hence, for security reason it is unacceptable as an authentication protocol for IP networks. Nevertheless, in view of its performance aspects the GSM protocol is an interesting candidate for presumably providing a fast key establishment.

In GSM the home network is represented by the *Home Location Register (HLR)*. The HLR is equivalent to the HA in MobileIP. In contrast to other approaches, the GSM key establishment is not initiated by the MN. The home network triggers handovers and message exchanges. The HLR is always aware of the MN’s point of attachment prior to the initiation of the key establishment.

The GSM protocol and MobileIP establish the session-key for different purposes. MobileIP desires authentication of the binding-update-messages. Hence, the session-key is used for the digitally signing of the authentication-header-extensions. The GSM protocol utilizes session-keys for encrypting the communication.

The IDKE and the GSM protocol both establish a key between the MN, the access network and the home network. The *GSM session-key* K_C corresponds to the IDKE's K_{SMS}. This is derived from a *Master Secret Key* K_I and the generation of a random number by means of a mathematical function. The protocol for exchanging this key between the MN and the network is illustrated in Figure 35. The message flow is as follows:

- (1, 2, 3) The HLR generates a random number and sends it to the MN. This random number acts as a challenge for the MN.
- (4, 5, 6) The MN must be aware of K_I and can thus create the session-key K_C. Based on K_C the MN is able to compute a response according to the received challenge. This response is sent back to HLR.
- (7,8) The HLR validates the response and sends an acknowledgment and keys to the *Base Transceiver Station (BTS)*. The access network allows the MN to access the network and to establish a connection.

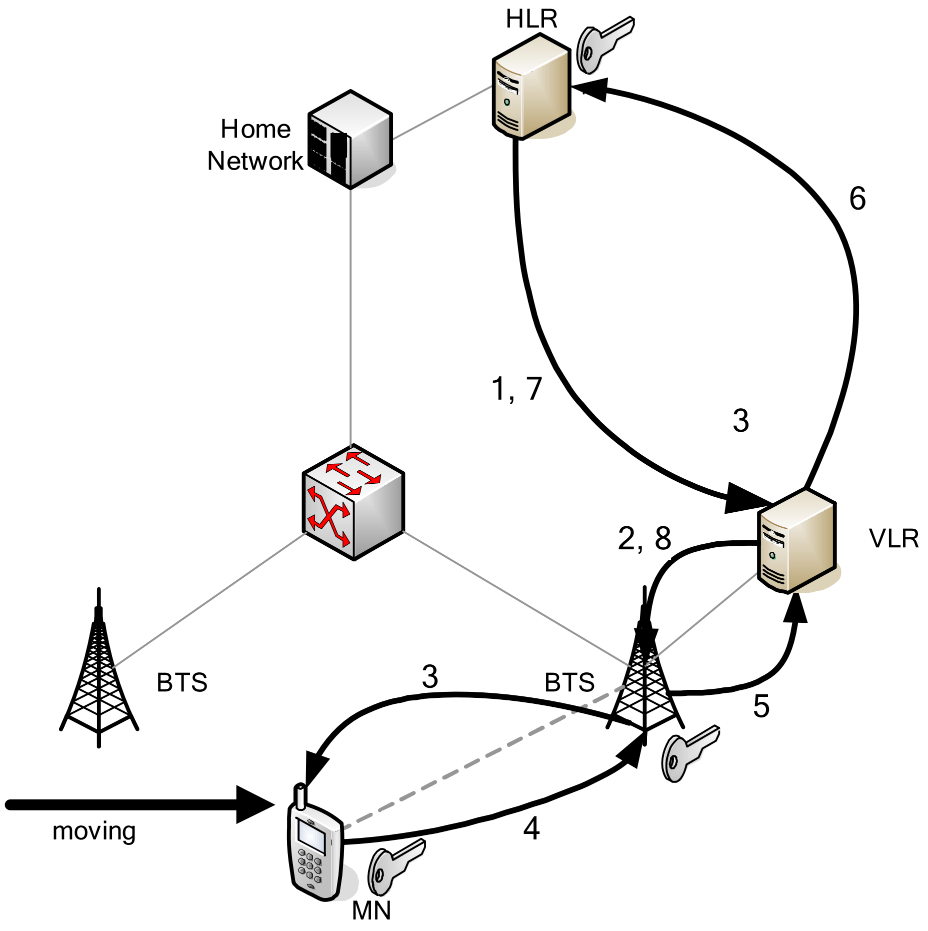


Figure 35: GSM Re-Authentication Message Exchange

6.2.2 Wireless-Shared Key Exchange (W-SKE)

The W-SKE protocol was especially developed for providing authentication and session-key establishment for MNs. The W-SKE protocol assumes an existing AAA infrastructure to set keys. It utilizes such widely used protocols as the *Extensible Authentication Protocol (EAP)* [ABV+04], *RADIUS* [RW00] and *Diameter* [CL03] for message transfers.

The W-SKE protocol minimizes the number of messages exchanged between the access network and the home network. The assumption is that the home network could be located faraway in terms of hops and delay. The W-SKE protocol only requires two messages between the access network and the home network. However, many messages are exchanged by means of a wireless link which could also prove to be a bottleneck.

In order to implement the IP based W-SKE protocol in the OPNET Modeler, some simplifications are necessary. The W-SKE requires protocols to be on top of the IP layer. The communication between the MN and the AR is encapsulated into an *EAP over LAN message (EAPOL)* [SH04]. EAPOL is a method of the *Extensible Authentication Protocol (EAP)* and was initially designed for LAN connections. The communication between the AR and the AAA servers utilizes the RADIUS protocol for exchanging messages. The performance simulation requires the focus to be on the message exchange, the packet size and the necessary computation time for authentication and key generation. The message flow is illustrated in Figure 36 and this functions as follows:

- (1) The MN sends a start message to the AR. This message contains the AR identifier. The start-message does not belong to the EAP protocol. EAP is always initiated by the core network. The start-message is used when clients need to initiate the message exchange.
- (2) The AR sends an EAP request message to the MN requesting the MN's identity.
- (3) The MN sends an EAP reply message to the AR containing the MN's *User Identifier (UID)* and the *Session Identifier (SID)*.
- (4) The AR forwards the message to the local *Foreign AAA (FAAA)* server. This server could be a RADIUS authentication server. The corresponding message is then a RADIUS authentication-request-message encapsulated into an EAP message.
- (5) The FAAA generates a random number N1 for each session and sends N1 back to the AR.

- (6) The AR forwards the message containing N1 to the MN.
- (7) The MN treats N1 as a challenge and creates an appropriate response for authentication purposes which is referred to as AUTH1. The MN also generates a random number N2. The MN sends a message to the AR. This message contains both N2 and AUTH1. AUTH1 is a message authentication code (MAC) and covers all the relevant data that is computed as AUTH1 = MAC $K_{MN,HAAA}$ (N1|N2|UID|SID|[ASID]). This is a keyed MAC based on the shared secret $K_{MN,HAAA}$ between the MN and the HAAA server.
- (8) The AR forwards AUTH1 to the FAAA server.
- (9) The FAAA server uses UID and SID to verify whether the MN has actually been attached to the AR. The FAAA sends a message to the HAAA of the MN. The connection between the FAAA and the HAAA server is assumed as being secure. Secure tunnels commonly exist between all AAA servers and AAA brokers.
- (10) The HAAA server validates the authenticity of the MN, based on the received UID. The HAAA computes an AUTH1′ based on the shared secret between the MN and the HAAA server which is similar to that of the MN. An AUTH1 and AUTH1′ should show the same value and therefore prove the MN's authenticity. The HAAA server generates a second authentication MAC termed AUTH2. The value is computed as AUTH2 = MAC $K_{MN,HAAA}$ (N2|N1|UID|SID|[ASID]). All single values are equal to AUTH1 but N1 and N2 occur in the reverse order. AUTH2 is designed to prove the authenticity of the HAAA server for the MN. The shared session-key, referred to as *Session Master Secret (K_{SMS})* is generated by the HAAA server based on AUTH2 and a *Pseudo Random Function (PRF)*. The session master secret is denoted as K_{SMS} = PRF $K_{MN,HAAA}$ (AUTH2). The HAAA server sends an AAA message to the FAAA server that contains AUTH2 and K_{SMS}.
- (11) The FAAA forwards the message to the AR.
- (12) The AR extracts the key K_{SMS} and AUTH2 from the AAA message and forwards AUTH2 to the MN, but not K_{SMS}. The MN itself computes the K_{SMS} based on the PRF, N1 and N2. Note that the key K_{SMS} is never sent via the wireless link.

The W-SKE protocol is based on two challenge response mechanisms. One is between the MN and the FAAA server, while the other is between the MN and the HAAA server. Hence, the W-SKE claims to provide mutual authentication for the MN and the access network.

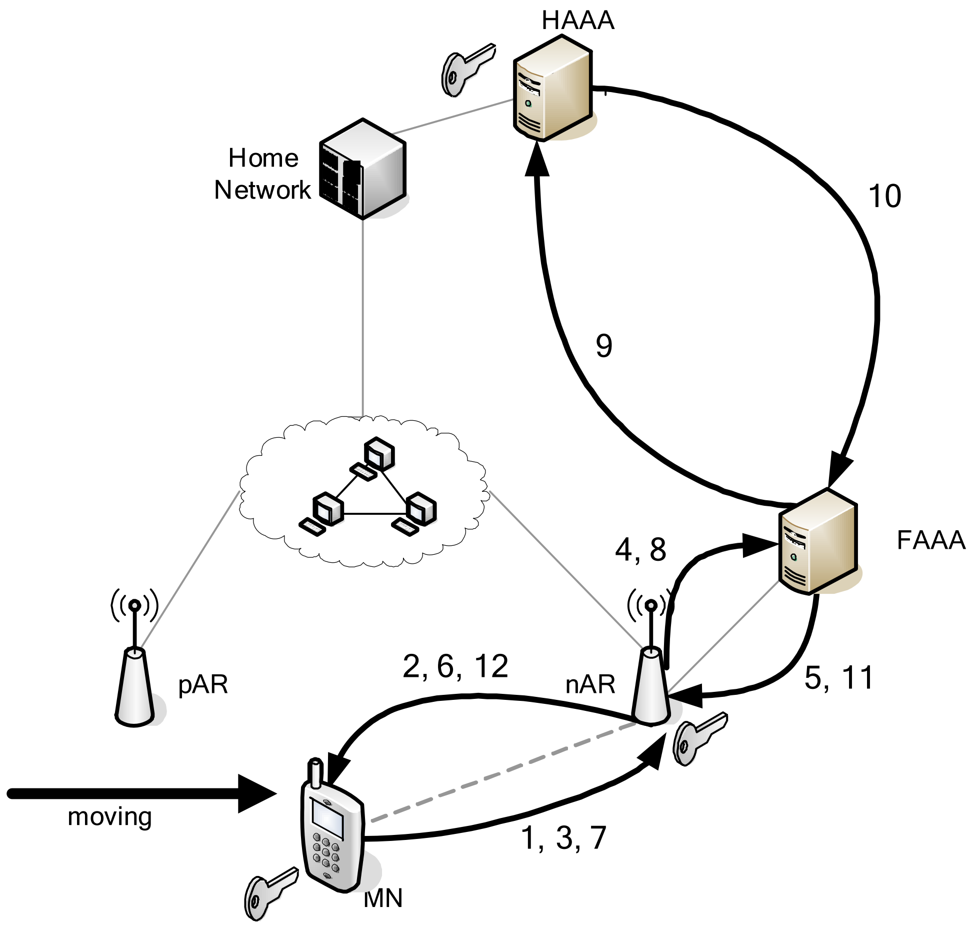


Figure 36: W-SKE Authentication Message Exchange

6.2.3 Simulator Implementation of the IDKE Protocol

The IDKE protocol specification has been described in detail in Chapter 3. Here, the IDKE protocol is modified in order to fit for the performance analyses. Two versions have been implemented: one with home network authentication and another without. The former is referred to as “IDKE with home authentication”, whereas the latter refers to “IDKE with temporary key establishment”.

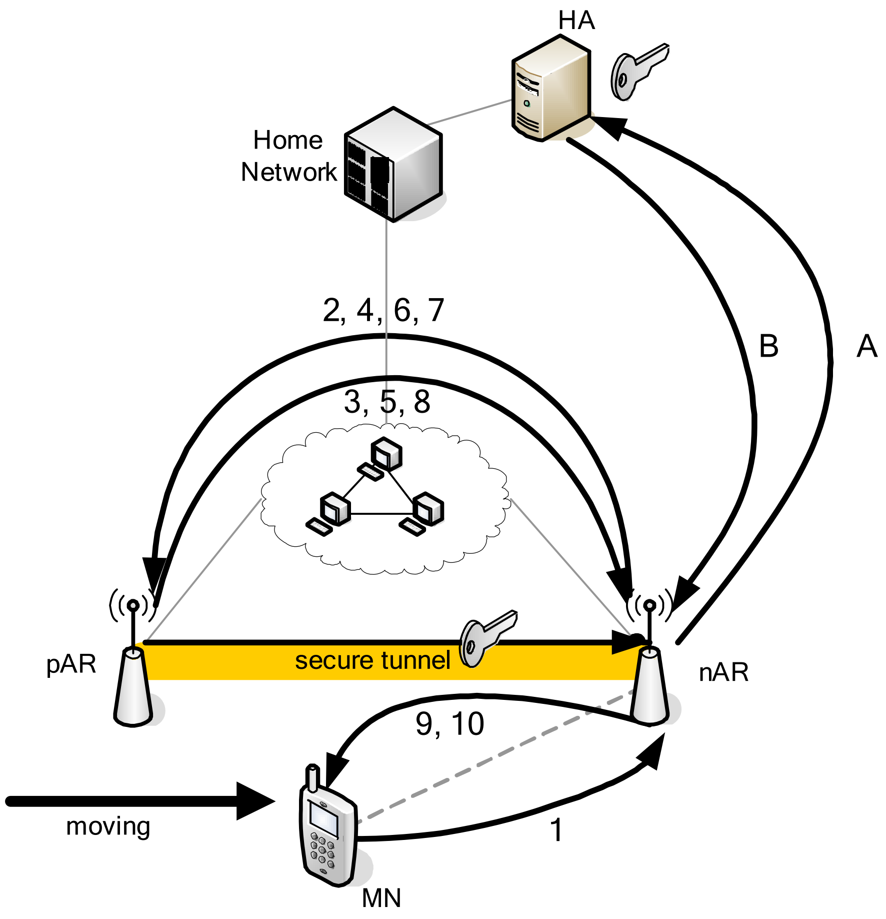


Figure 37: IDKE Re-Authentication Message Exchange

Both of these versions cover messages 1 - 10 as illustrated in Figure 37. The Message *A* and *B* are only used in the home authentication version. In contrast to the IDKE specification in Chapter 3, the first message is sent from the MN to the nAR. The reason for this is that the router advertisement is not considered in the performance analyses. It is assumed that prior to the desire for a handover, an advertisement has already been received. The MN's handover request-message identifies the starting point of the handover procedure. The last message is removed that is, for authentication purpose, sent by the MN to the nAR. Thus, the nAR finally sends two messages to the MN, one to acknowledge the temporary key which confirms the key forwarding and another message for a successful home authentication. Therefore, the modified message flow works as follows:

- (1) Token request message to initiate both the key transfer and the home authentication.
- (2, 3) The IDKE token forwarding and acknowledgement.
- (4, 5, 6) A Diffie-Hellman [BM99] based key exchange, such as the IKE [Bor00] in aggressive mode.

- (7, 8) Context Transfer Protocol (CxTP) [LNP+05] utilization for the request for and the transfer of data, as well as that of the session-key.
- (9) Notification of the MN that the key has been transferred successfully.
- (A, B) Home authentication procedure.
- (10) Notification that the home authentication has been successfully performed.

6.3 Simulator Setup

The discrete event simulator OPNET Modeler is used for evaluating the performance of all protocols. The topology is implemented as a general scenario that describes an MN performing a handover between two ARs. The protocols are specified in the simulator's language and include the probes for estimating any delays. In order to find an optimal environment for each protocol, all of the protocols have been simulated under a variety of different conditions.

Therefore, the MN always performs the same handover, whereas the conditions are permanently changing. The delay on each link is changed separately in order to identify the influence on the protocol performance. A summary of the results is presented in a graph. The complete protocol results have been merged, thus enabling the protocols to be compared under different conditions. The simulation deployment process is introduced at first, followed by the conclusions gained from the results. The necessary design choices are also explained.

6.3.1 Packet Formats

Packet formats are designed for any messages pertaining to protocols. All packets are somewhat smaller than in real implementations. The size of all identifier fields is set to 16 bits. However, in conforming to IPv4 or IPv6 implementations, the address-fields have to be 32 bits, respectively 128 bits in size. The IPv4-conform packet formats are presented in Appendix B. The following additional simplifications have been made in order to produce proof of concept:

- Fields for tokens, keys and encrypted data are initialized by random values.
- Timestamps and nonces are set to fixed values.
- Session identifiers are set to a fixed value.

These simplifications have been introduced in order that the protocol implementation remains uncomplicated. The simulations evaluate a single MN by considering a single session under an immense variety of conditions. Malicious nodes have not been considered within the performance evaluation. All cryptographic operations such as token generation, encryption, decryption, hash computation and key generation have been modeled as processes that cause a delay prior to setting any values to the messages. The message size also has an influence on the performance. The following messages have been defined for the IDKE protocol:

- ***idke_initiation:*** A 180 bits message to be sent from the MN to the nAR in order to initiate the key transfer.
- ***idke_forward:*** The message includes the entire content of the *idke_initiation* message. This 212-bit message is sent from the nAR to the pAR.
- ***idke_acknowledge:*** This is a 180-bit message that is sent back from the pAR to the nAR.
- ***idke_ike:*** This message is sent 3 times between the nAR and the pAR. The purpose is to establish the shared key for tunneling. The message size is 212 bits.
- ***idke_ctp_request:*** This 260-bit message is sent from the nAR to the pAR for the request of the session-key.
- ***idke_ctp_reply:*** This message carries the key from the pAR to the nAR and it corresponds to the CxTP data message. Due to certain fields which have been reserved for later purposes, the size is set at 404 bits.

Messages that do not carry any specific data have been set at 212 bits. These standard messages are: *idke_tmp_key_established*, *idke_home_request*, *idke_home_reply* and *idke_accomplished*.

The following messages have been designed for the GSM and the W-SKE protocol:

- ***gsm_packet, wske_packet:*** This message is used for the GSM and the W-SKE protocol to transport challenges and responses. The size is set at 244 bits. These packets are used for the GSM and W-SKE messages that do not transfer any key material.
- ***gsm_packet_key, wske_packet_key:*** This message is used whenever a key is transferred. The size of this message is 372 bits.

Examples of how messages appear in the packet format editor are given in Figure 38 and Figure 39. The former represents the *idke_forward* message, while the latter illustrates the *idke_ctp_reply* message.

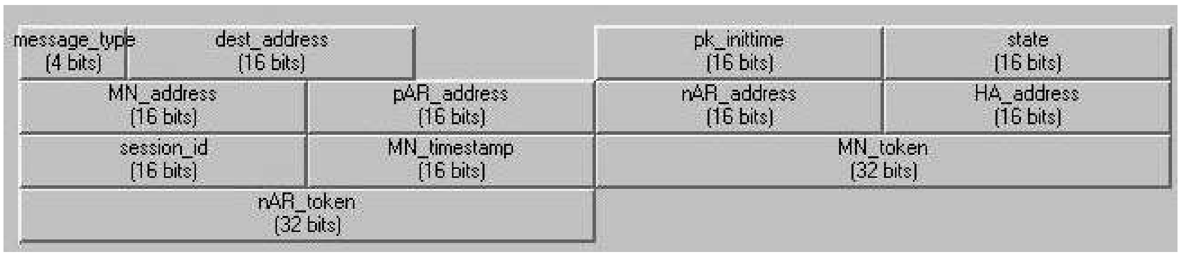


Figure 38: The *idke_forward* Packet in the OPNET Packet Modeler

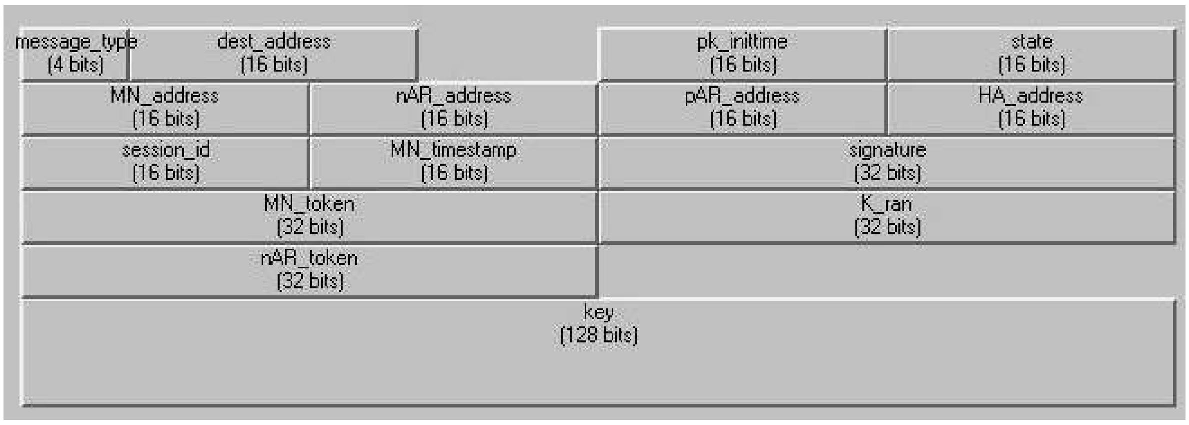


Figure 39: The *idke_ctp_reply* Packet in the OPNET Packet Modeler

6.3.2 Nodes and Links

A network consists of nodes and links. Nodes are for instance routers, hosts and servers. Links interconnect nodes and provide communication channels. They can either be wired or wireless.

Node models describe the internal structure of nodes. This structure usually consists of one or more protocols and streams. Streams act as interfaces to links which are outside the node model. Protocols are either directly connected to streams or to other protocols.

Three new node models are specified in each protocol for the purpose of evaluating performance:

- ***MN node model:*** The MN includes the protocol and two network interfaces as illustrated in Figure 40. The network interface is represented by two streams; one for incoming and one for outgoing traffic.
- ***nAR node model:*** The nAR requires two network interfaces for routing purposes. The node model is illustrated in Figure 41. The two network interfaces are represented by four streams.

- ***pAR and HA node model:*** The same network model is used for pAR and HA since both act as responders to the key request. If the pAR is a router, it's node model is similar to that of the MN's. Neither the HA nor the pAR need to provide a routing function in this simple scenario.

Protocols within a node model are represented by a "grey box" in order to conceal internal actions.

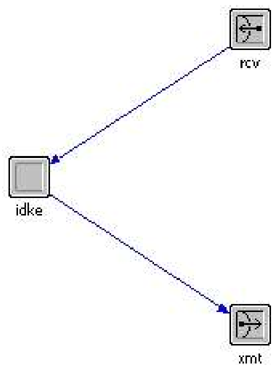


Figure 40: Node Model for pAR, HA and MN

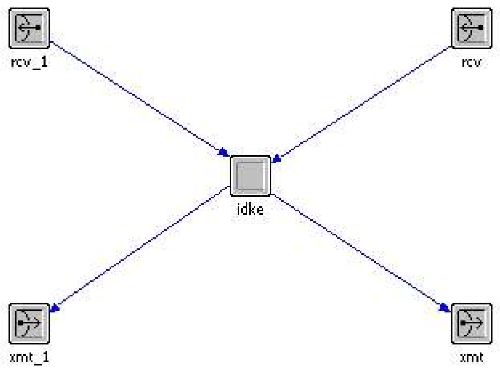


Figure 41: Node Model for nAR

6.3.3 Processes

Each protocol is considered as a process. The OPNET Modeler provides a tool for the specifying of processes as a finite state machine. Thus, a ***process model*** specifies and defines the handling of requests, challenges and the key-forwarding. The process model also simulates delays according to the key computation and encryption.

Process models are created for each protocol in order to evaluate the performance. The IDKE protocol for example, has process models that depend on a node:

- The MN, presented in Figure 42, initiates the key request and waits for a response. A probe captures the time and stores the delay for further analyses.
- ARs provide a more complex function. The nAR for example, shown in Figure 43, includes a state for a key negotiation in order to cause delays.

Objects within a process model are referred to as states that are interconnected by links. Each link corresponds to a procedure. This procedure is executed when leaving a state via this link.

Nine new process models are involved in the performance simulation. There are three new models for each protocol where the HA and the pAR use the same state machine. The new state machines and their purposes are summarized in Table 6.

IDKE	GSM	W-SKE
HA, pAR (responder)	HLR (responder)	HAAA (responder)
nAR (forwarder)	nAR (forwarder)	nAR (forwarder)
MN (initiator)	MN (initiator)	MN (initiator)

Table 6: New OPNET Process Models

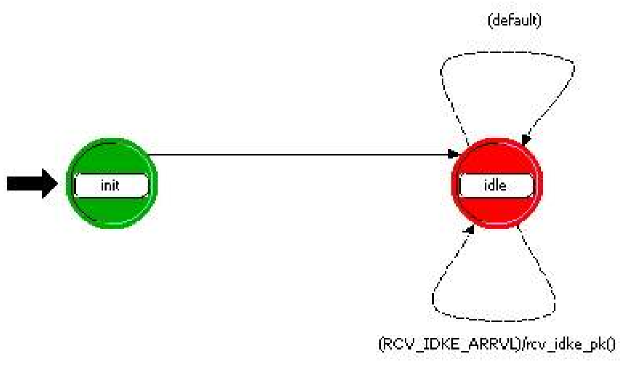


Figure 42: The IDKE MN Process Model

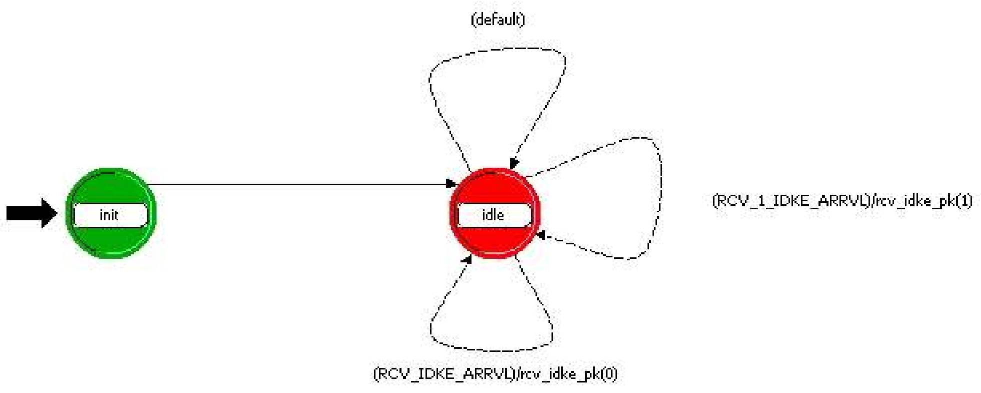

(default)
init
idle
(RCV_1_IDKE_ARRVL)/rcv_idke_pk(1)
(RCV_IDKE_ARRVL)/rcv_idke_pk(0)

Figure 43: The IDKE nAR Process Model

6.4 Simulation Scenario

The network topology consists of one MN, two ARs and one HA as illustrated in Figure 44. The ARs and the HA are directly connected to the core network represented by an IP cloud. The MN is attached to the nAR via a wireless link. The focus is on calculating the overall handover delay. Therefore, several paths and corresponding delays have been defined. The paths between the nodes are represented by X, Y, Z and α. The definitions are as follows:

- $\boldsymbol{\alpha}$ is the path that connects the nAR to the core network. When the nAR sends packets, the initial part of the path is the same independent of whether the packets are sent to the pAR or to the HA. This common path is referred to as α. This path delay covers latencies caused by intermediate routers.
- $\boldsymbol{X}$ represents the wireless link between the MN and the nAR. X covers the whole transmission latency that occurs on the wireless link.
- $\boldsymbol{Y}$ is the path between the pAR and the core network. This is where the path of α ends. Both α and Y together, describe the path between the pAR and the nAR. However, any delays caused by the core network in transit between the domain A and B, belong to the path Y.

$\boldsymbol{Z}$ is the connection for the MN's Home Network up to the matching point of Y and α. It is possible that the home network is located at a distance in regard to hops and thus the delay could be somewhat higher.

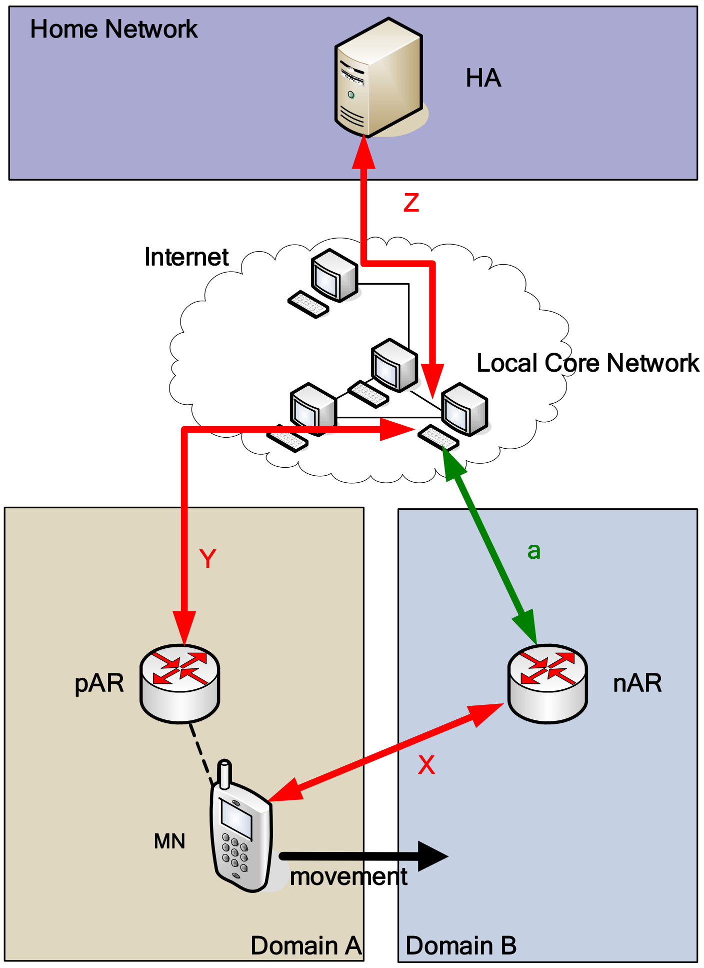


Figure 44: Network Topology

The assumption in terms of delay is that all paths have equal return paths. The paths *α*, *Y* and *Z* are assumed to share one endpoint. This assumption is reasonable, owing to the fact that no data is sent from the pAR to the home network during the handover procedure. The common link *α* is assumed to provide anything but fixed quality, whereas the other paths vary during the simulation process. In order to reduce the amount of influencing parameters, *α* can model any path delay which is fixed in quality during one simulation run. Hence, *α* is a parameter which belongs to the environment and can either provide low or high latencies, whereas *X*, *Y* and *Z* are dependent on the network conditions. Therefore, *X*, *Y* and *Z* are variables that change during the simulations. The aim is to find a method for estimating the influence of the path delays of *X*, *Y* and *Z* on the overall performance of each protocol.

6.5 Simulation Results

The simulation results are dependent on the path quality. Therefore, variations of X, Y and Z have an impact on the protocol performance. In order to discover any quality dependences, each variable is evaluated separately. Other variables have also been fixed and these describe the setting. This setting can either describe a *high quality (HQ)* environment where delays are reasonably low on other links; or the setting describes a *low quality (LQ)* environment where the delays are respectively high. Consequently, this requires six test cases where there are two settings for each variable. The protocols are evaluated parallely in every test case. The network architecture for the OPNET Modeler is illustrated in Figure 45.

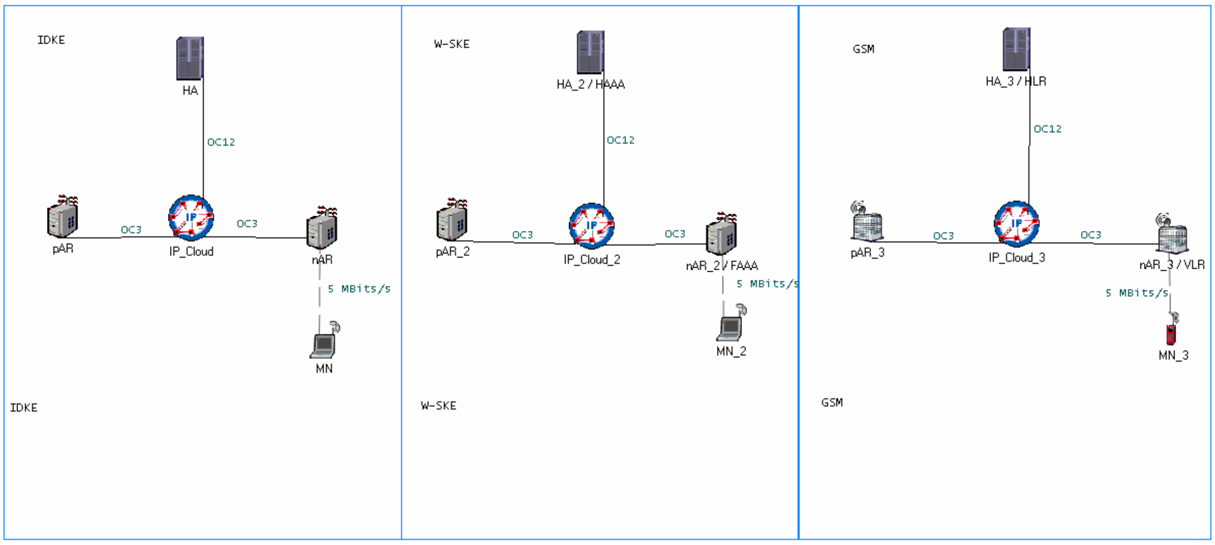


Figure 45: Topology in the OPNET Modeler

6.5.1 Home Link Analyses

The MN performs a handover between two neighboring ARs. However, the MN's Home network can be located anywhere in the network. The home network can either be one of the domains which the MN roams in between or be located faraway. The delay to reach the home network ranged between 0 and 250 ms during the course of the simulation. The core network and the wireless link permanently provide low latencies in this evaluation. The overall protocol handover delays have been measured and are depicted as a graph in Figure 46. The x-axis represents the delay to the home network, whereas the y-axis shows the corresponding handover delay which can be up to one second. Protocols are illustrated in different colors in the graph.

(The wireless link delay is 10 ms; the core network delay is 5 ms.)

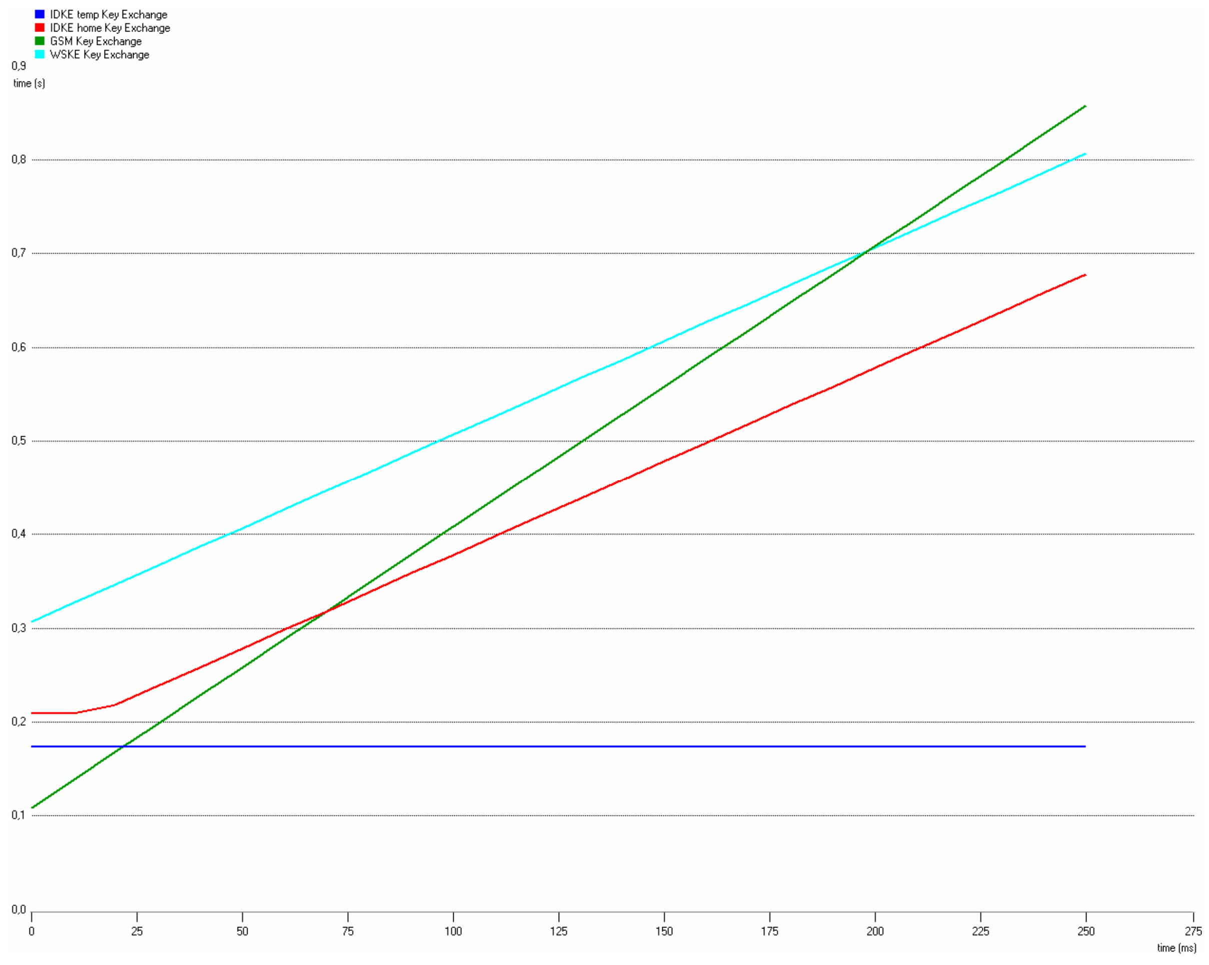


Figure 46: Handover Delay – Link to the Home Network Variable – Core and Wireless HQ

Figure 46 shows that the GSM protocol is the fastest approach when the home network delay is less than 20 ms. The IDKE temp is the fastest protocol in cases where the home network delay is higher than 20 ms. The graph illustrates the IDKE_temp as a constant value since it is entirely independent of the home link. However, the IDKE_home is faster than the GSM protocol when the home link delay is higher than 70 ms. The W-SKE protocol is constantly slower than both of the IDKE approaches in this scenario. Nevertheless, the W-SKE is faster than the GSM protocol when the home link delay is over 190 ms.

These considerations assume low quality (LQ) on the wireless link (delay of 30 ms). This is due to retransmission and package loss as well as in the core network (which also has a delay of 30 ms). Figure 47 shows a graph for the LQ-scenario. If the delay on the home link is at the most 85 ms, the GSM approach provides the fastest handover. When the delay to the home network is higher than 85 ms then the IDKE_temp is faster than the GSM.

The W-SKE and the IDKE_home show equivalent dependencies under these conditions.

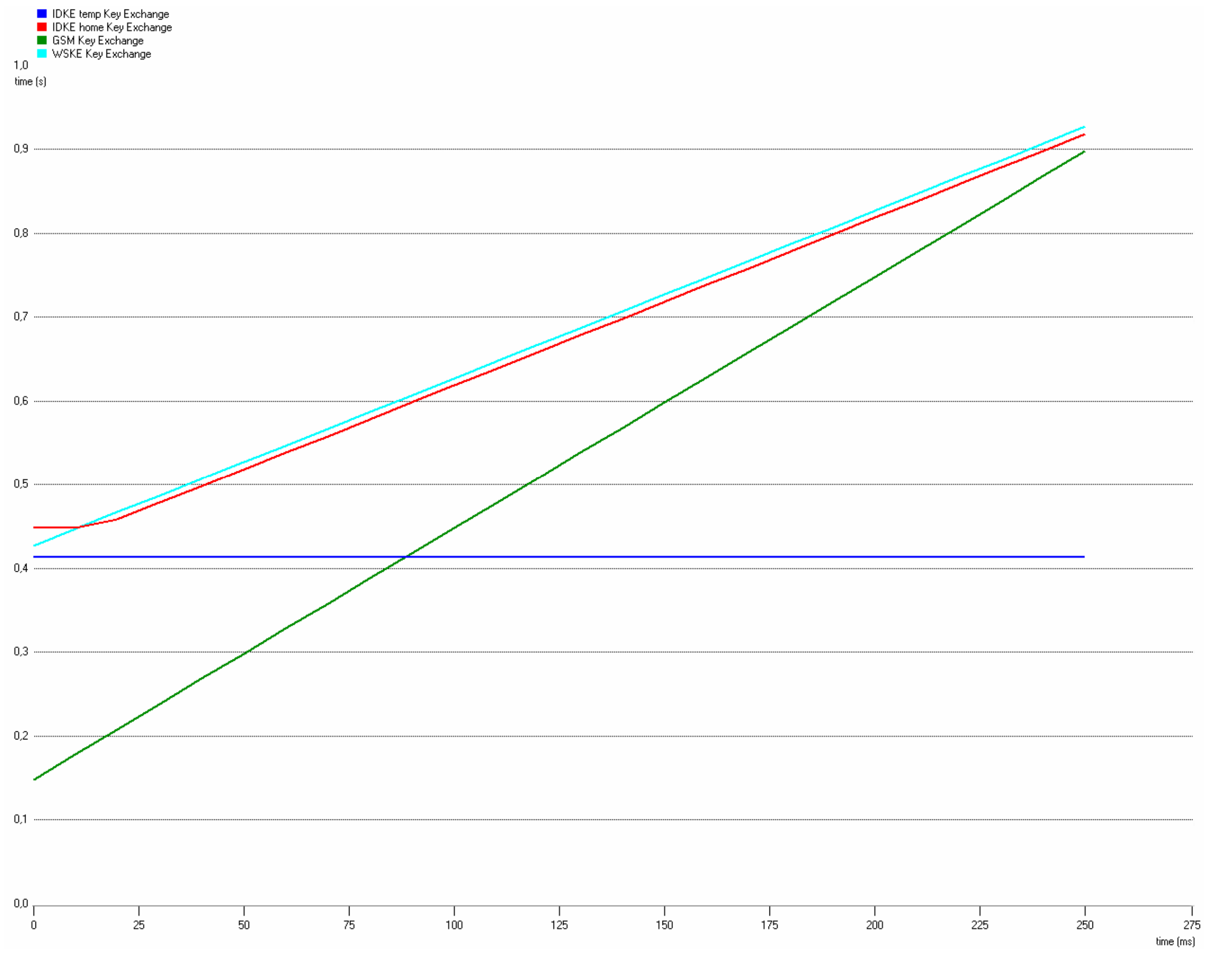


Figure 47: Handover Delay – Link to Home Network Variable – Core and Wireless LQ

6.5.2 Wireless Link Analyses

The delay on the wireless link may vary due to some congestion caused by a high amount of participants, packet loss and retransmissions. In this setup, the delay on the wireless link varies between 0 ms and 50 ms.

The first setup assumes high quality in the core network (5 ms delay) and for the home link (30 ms delay). Figure 48 illustrates the results for these protocols. This setup depicts an environment that is similar to that of an intra-domain handover. The home network is very close in terms of the delay and there is also a fast communication between the AR.

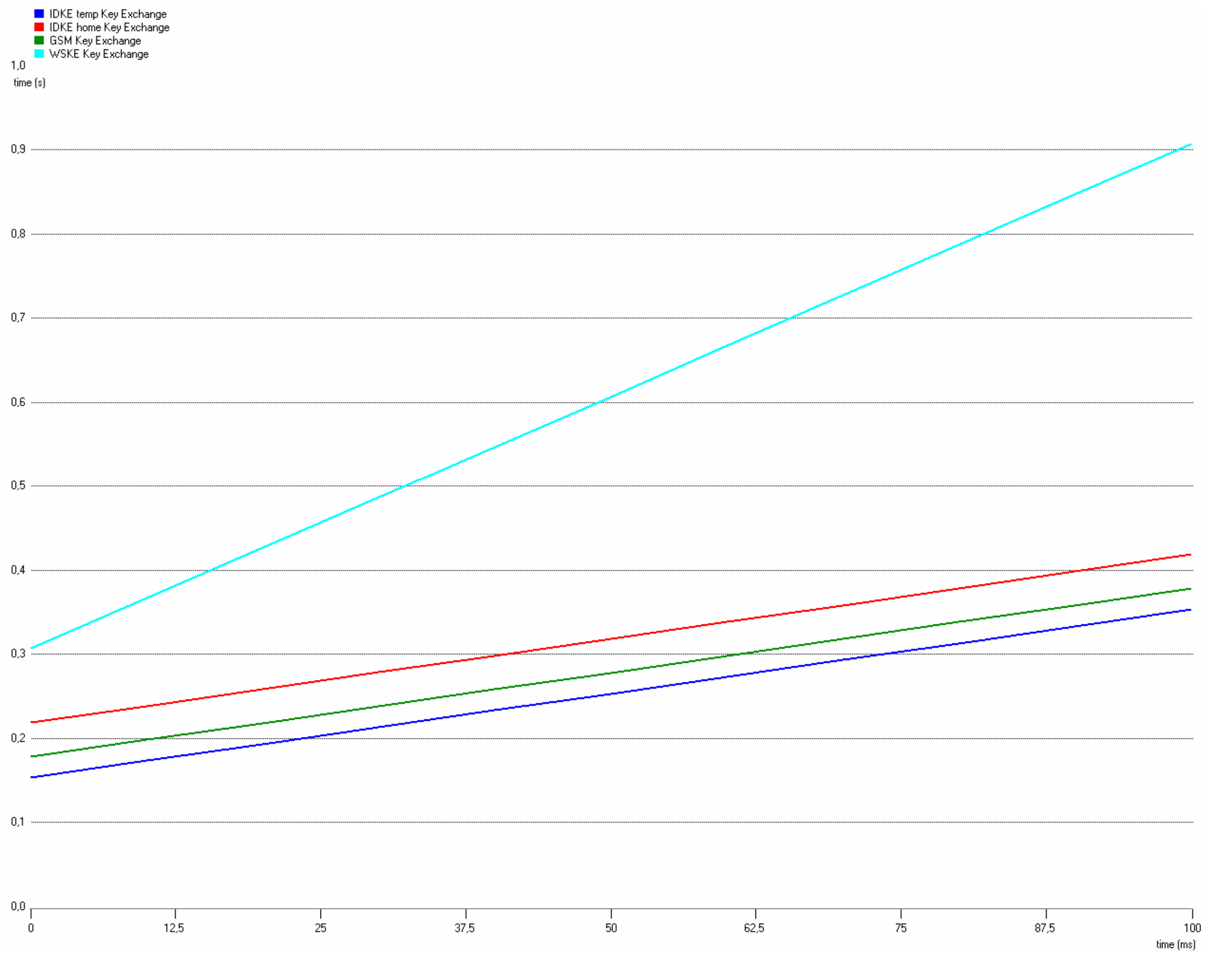


Figure 48: Handover Delay – Wireless Link Variable – Core and Home Link HQ

Figure 48 shows that the IDKE_temp is faster than the GSM protocol. However, the IDKE_ home is slower than the GSM protocol. The GSM protocol and both of the IDKE protocols, are less dependent on the wireless link than the W-SKE protocol. The graph shows that the W-SKE protocol increases faster than any other protocol. This is due to the high amount of messages exchanged over the wireless link. Thus, this test run shows that both of the IDKE protocols provide a high performance even when performing intra-domain handovers.

A setup of low quality in the core network and on the home link is examined in the LQ-scenario. The core network delay is 30 ms while the home link delay is 100 ms. Figure 49 shows that the IDKE_temp is faster than the GSM protocol. However, the IDKE_ home is slower than the GSM protocol. The GSM protocol and both of the IDKE protocols, are less dependent on the wireless link than the W-SKE protocol. The graph shows that the W-SKE protocol increases faster than any other protocol. This is due to the high amount of messages exchanged over the wireless link. Thus, this test run shows that both of the IDKE protocols provide a high performance even when performing intra-domain handovers.

A setup of low quality in the core network and on the home link is examined in the LQ-scenario. The core network delay is 30 ms while the home link delay is 100 ms.

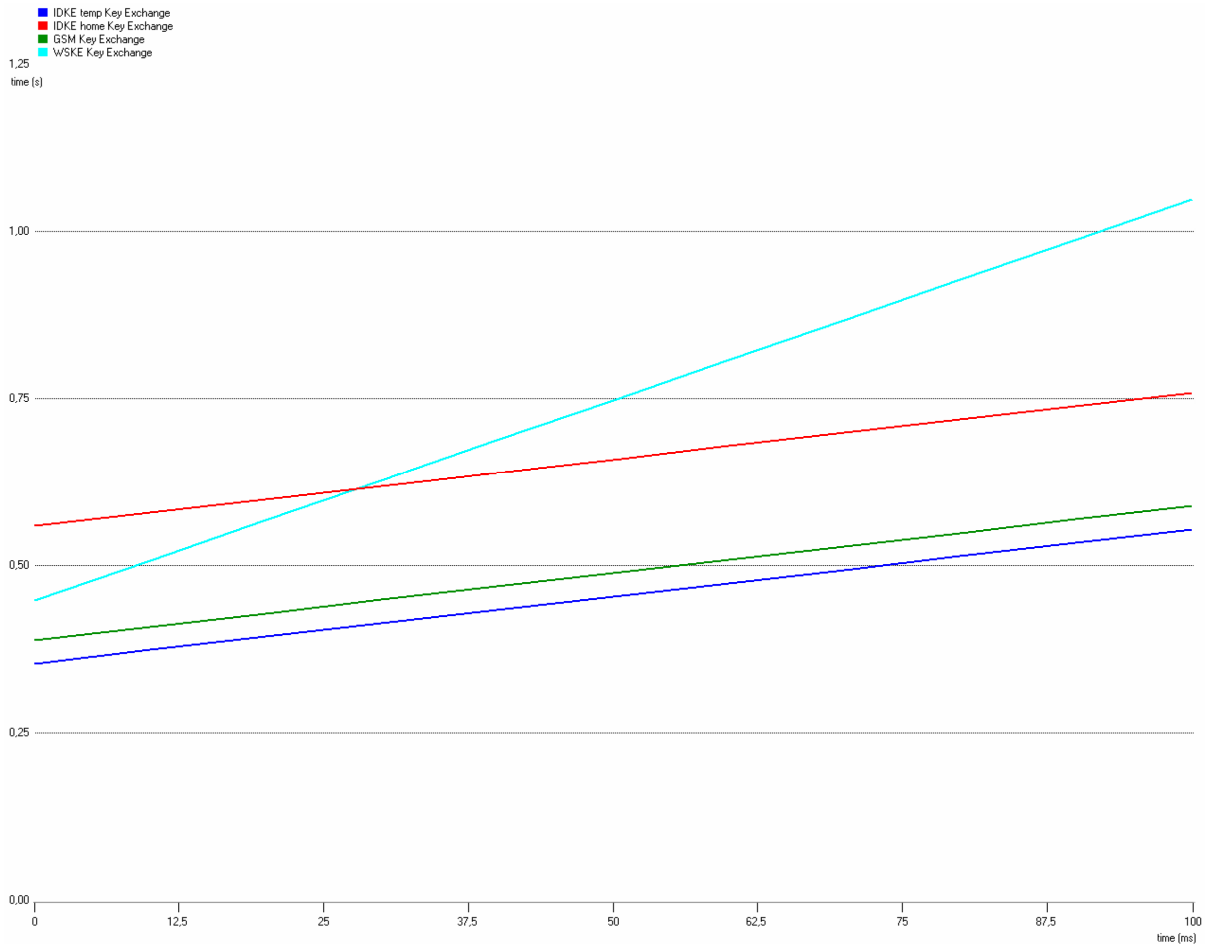


Figure 49: Handover Delay – Wireless Link Variable – Core and Home Link LQ

6.5.3 Core Network Analyses

The MN performs the handover between two topologically closely located domains. However, it is possible that the performance in the core network may not be constant. Thus, the influence on the quality caused by the path between the ARs has been examined in two further scenarios. In the first setup, the wireless link provides a high performance (delay of 10 ms) and there is a high performance on the home link (delay of 50 ms).

The results are presented in Figure 50. This graph shows when the core network delay is less than 8 ms, the IDKE_temp is faster than the GSM protocol. If the core network has a delay of more than 8 ms, then the GSM protocol is the quickest protocol means.

The IDKE_home is faster than the W-SKE when the core network delay is less than 20 ms. The IDKE_temp is even swifter in cases where there is a core network delay of less than 28 ms. Hence, the GSM protocol is the fastest protocol. However, if the core network delay is around 40% or 60% of the home link delay, then one of the IDKE protocols is quicker than the W-SKE.

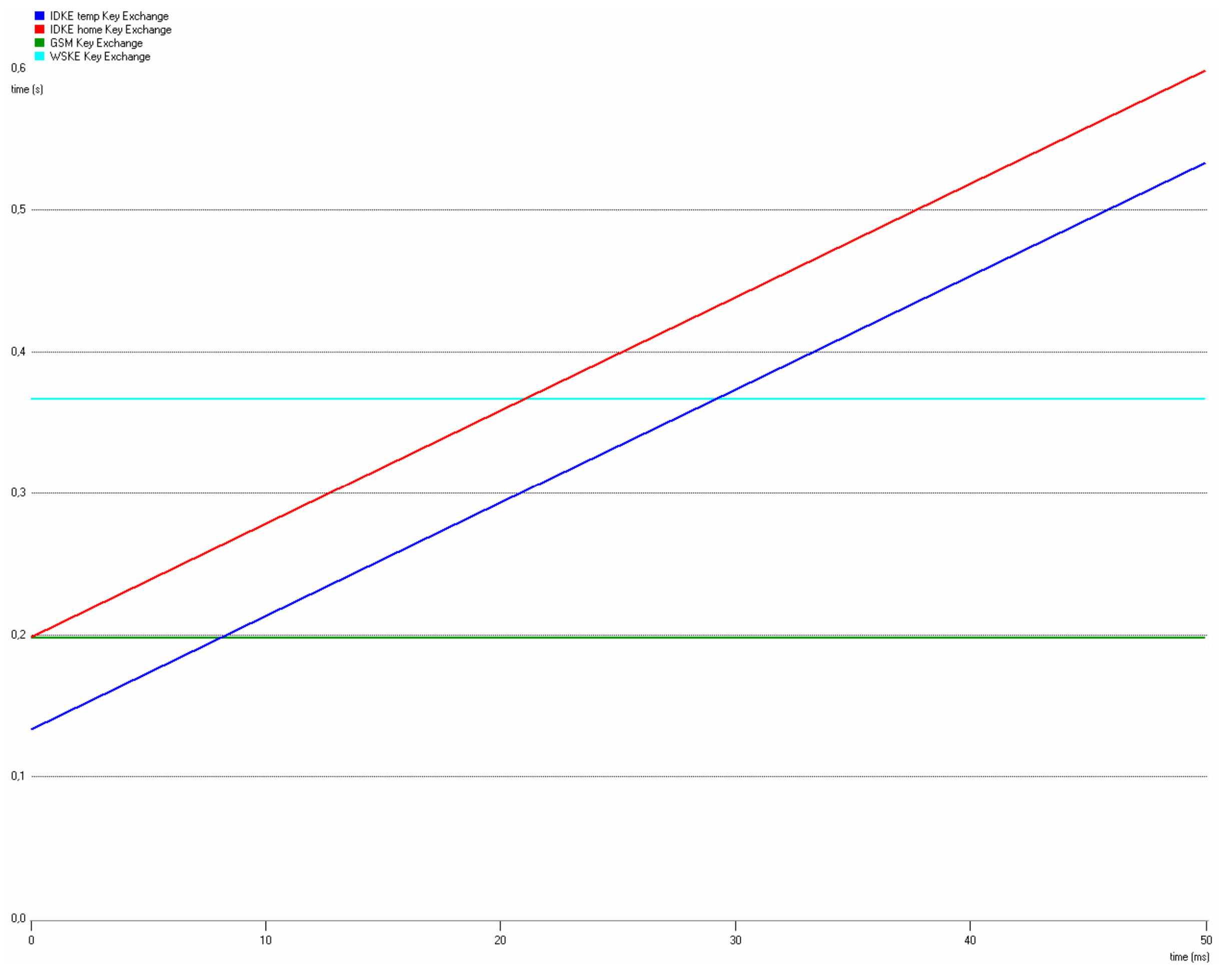


Figure 50: Handover Delay – Core Network Delay Variable – Wireless and Home Link HQ

It has been assumed in the following that there is low quality on the wireless link (delay of 30 ms) and on the home link (100 ms) as well as that the core network delay varies between 0 ms and 50 ms.

The results are presented in Figure 51. When one compares the HQ- and LQ-scenario, it can be seen that the two lines of the IDKE protocols run in parallel at both scenarios, but that there is a higher margin in the LQ-scenario. The GSM and the W-

SKE protocol are represented as constants in both scenarios as both are independent of the inter AR communication. In Figure 51, tthe intersection occurs later than in the first scenario. This illustrates that the delay in the IDKE protocols is much lower than the delay in the other protocol. The IDKE_home is faster than the W-SKE protocol, if the core network delay is less than 32 ms. It is even faster than the GSM protocol when the core network delay is less than 9 ms. The IDKE_temp is always quicker than the W-SKE protocol, if the core network delay is less than 50 ms. It is also swifter than the GSM protocol in cases where the core network delay is less than 34 ms.

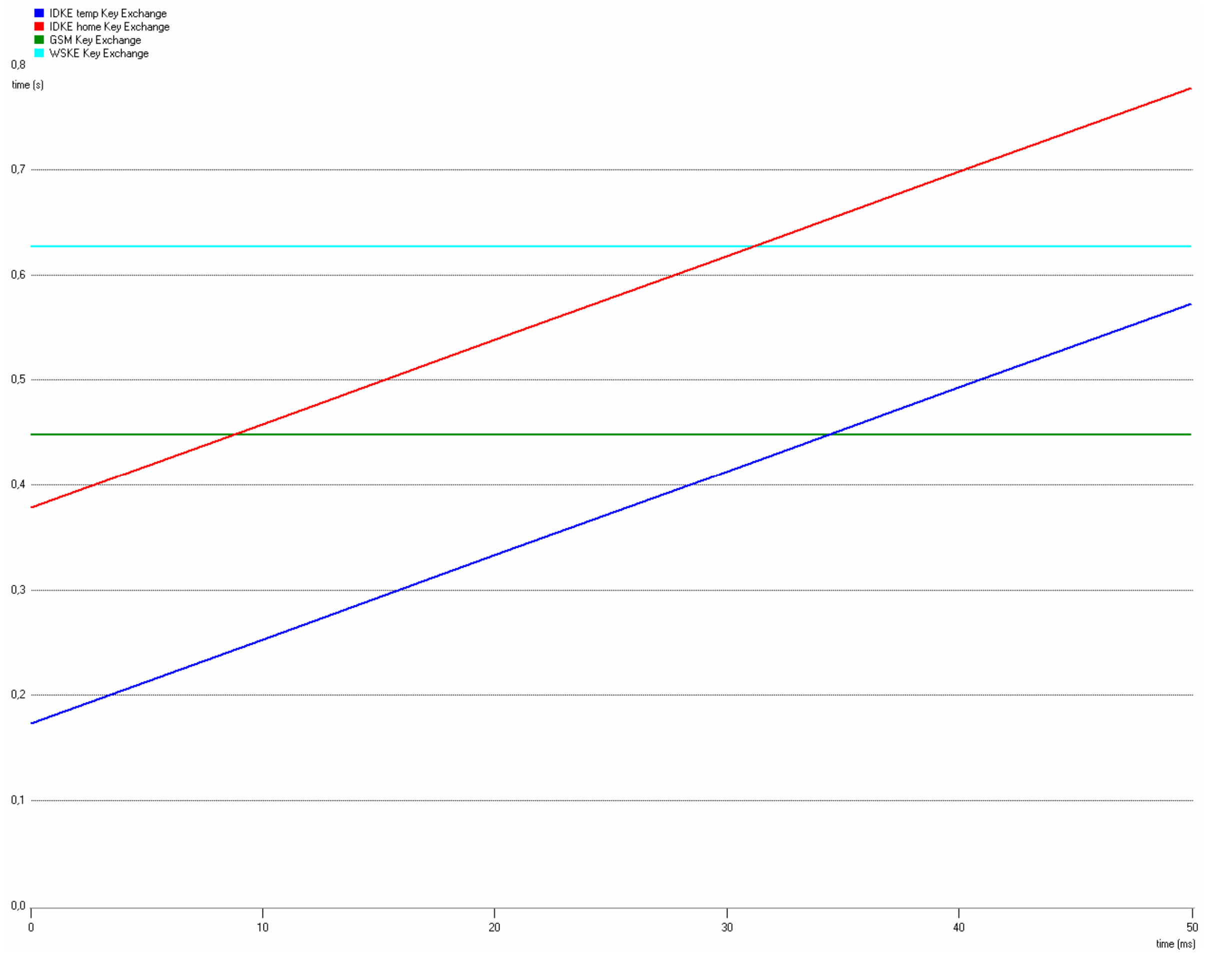


Figure 51: Handover Delay – Core Network Delay Variable – Wireless and Home Link LQ

6.6 Summary, Discussion & Outlook

Several protocols have all been simulated in a variety of different conditions. Therefore, the simulation results imply that the conditions under which each protocol has been subjected to, should provide the best performance. Simulations have been performed on the GSM, the W-SKE and the IDKE protocol with a temporary key establishment as well as the IDKE protocol with a home authentication procedure. The dependence has been analyzed for all four of the protocols taking into consideration the quality of the wireless link, the home link and the core network quality. The results are summarized in Table 7.

Quality	GSM	W-SKE	IDKE temp key	IDKE home key
core network	independent	independent	highly dependent	highly dependent
LQ	> 34 ms fastest	> 20 ms faster than IDKE home key	< 34 fastest	< 9 ms faster than W-SKE, GSM < 32 faster than W-SKE
HQ	> 8 ms fastest	> 30 ms faster than IDKE temp Key	< 8 ms fastest < 28 ms faster than W-SKE	< 20 ms fast, faster than W-SKE >32 slowest
home link	dependent	dependent	independent	dependent
LQ	< 85 ms fastest	> 10 ms slowest	> 85 ms fastest	slow equal to W-SKE
HQ	< 20 ms fastest	< 190 ms slowest	> 20 ms fastest	> 70 ms faster than W-SKE, GSM
wireless link	slightly dependent	highly dependent	slightly dependent	slightly dependent
LQ	Fast	slowest (extremely slow)	fastest	fast
HQ	equal to IDKE_home_key	slowest	always faster than GSM	fast equal to GSM

Table 7: Dependency on the Link Quality

Obviously, the IDKE with a home authentication will always be slower than the IDKE protocol with a temporary key establishment as the home authentication implies a temporary key transfer as the former implies the latter. Table 7 shows that the IDKE protocol is highly dependent on the core network quality. The IDKE is the fastest protocol when the core network delay is at the most 8 ms. If the home link or the wireless connection provide low quality, then the IDKE offers the best quality for all protocols if the core network delay is at least less than 34 ms. This will most likely be the case in the core network. The IDKE with a home authentication is the fastest protocol, if the core network delay is less than 9 ms.

GSM is a fast protocol that provides a quick key exchange when the home network and wireless connection provide HQ. This can be seen in the last two lines of Table 7. The GSM protocol is the fastest protocol in the HQ-scenario, assuming that the core network provides relatively low quality. It is always the quickest, if the home link is also fast < 85 ms (LQ); < 20 ms (HQ). This fact also implies that the GSM protocol is the fastest when the MN is near to the home network. Thus, whenever entering or leaving the home domain the GSM is expected to be the fastest protocol.

The IDKE with a temporary key establishment is entirely independent of the home link delay and is therefore robust against a broken link.

The analyses on the dependency of a wireless link showed that the GSM and the IDKE protocol are not strongly dependent on the quality of a wireless link. In contrast, the W-SKE protocol is highly dependent on it. Thus, it can be seen that the W-SKE protocol has been developed for a high quality wireless link and for functioning at long distances from the home network.

In conclusion, it has been shown that all protocols have their strengths under various conditions and it is most likely that all of them will be able to coexist. The best approach might be to merge them together and run them in parallel to each other in order to provide a maximum quality for all circumstances. Running evaluations on this merged protocol might be interesting for future studies. Further investigations could be done in integrating the IDKE protocol in a MobileIP environment in order to test the performance. Simulations on a huger amount of moving MN is also a potential candidate for further studies on order to evaluate the scalability of the IDKE protocol.

Chapter 7 Conclusions

This thesis has introduced, specified and verified the Inter-Domain Key Exchange Protocol (IDKE) under various aspects relating to security, robustness and performance. The task of the protocol is to establish a session-key at the new point of attachment, especially after an MN has performed an intra-domain handover. Consideration has been given to existing mechanisms and other current approaches. Formal analysis and simulations have been performed and the results were used to extend the protocol specification in order to successively improve the capabilities of the protocol.

Security verifications have been performed for the purpose of proving that the IDKE protocol fulfills secrecy and authentication properties. Supplementary analyses were used to remove unnecessary components of the protocol in order to obtain a lightweight protocol that can be processed at speed. The utilized model checking approach named Casper/FDR proved to be capable of this task. Details on security, the evaluation and subsequent work on security verification can be found in the summary in Section 4.4.

The robustness analyses focused on specifying and extending the IDKE-AR by using the SDL and simulating the behavior of a number of ARs by means of MSCs. The extended IDKE protocol specification proved to be capable of handling concurrent protocol runs. The extension covers cancel messages and the necessity of timestamps carried by some messages. Details on the extension, its capabilities and future work in this area are given in the summary in Section 5.5.

The third section of this thesis describes the performance evaluations that act as a proof of concept on the IDKE protocol. The aim of this study was to investigate as to whether the IDKE protocol is able to provide better performance than other approaches. By using the discrete event simulator OPNET Modeler, the IDKE protocol was compared with the GSM and the W-SKE. It could be shown that under certain circumstances, especially when running under inter-domain-handover conditions, the IDKE protocol is expected to be faster then the other approaches. More details containing the performance evaluation and a table summarizing all information can be found in the summary in Section 6.6.

In conclusion, it can be stated that by providing localized inter-domain handover function the IDKE protocol provides a possible solution for an open issue of the CxTP. The combination of the CxTP and the IDKE protocol can be used to re-establish session-keys, but is not limited to being used in 802.11 *Wireless Local Area Networks (WLANs)*. The IDKE protocols can also improve the handover capabilities in future mobile environments based on the 802.20 *Mobile Broadband Wireless Access (MBWA)* or the 802.16 *Worldwide Interoperability for Microwave Access (WiMAX)*.

It can be further stated that the methodology of combining security, robustness and performances evaluations by means of formal methods has improved the IDKE protocol. This methodology may be useful for other security protocols that are also required to provide a high performance. Thus, this combination of formal verification tools has formed the basis for ultimately obtaining a fast and robust protocol that can also fulfill all the desired security properties. However, the main drawback of this methodology is that it involves many different tools. Thus, the particular protocol under evaluation will always need to be specified from the outset as all of the tools and mechanisms require different representations, specifications, or implementations. This prompts the desire for an integrated general evaluation environment capable of verifying a vast variety of protocols from a single specification.

Bibliography

[ABV+04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, *Extensible Authentication Protocol (EAP)*,RFC 3748, Internet Engineering Task Force (IETF), Network Working Group, June 2004.

[ACG+84] W. Alexi, B.-Z. Chor, O. Goldreich, C.P. Schnorr, *RSA and Rabin Functions: Certain Parts Are as Hard as the Whole*. Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, pages 449-457, 1984.

[ACG+88] W. Alexi, B.-Z. Chor, O. Goldreich, C.P. Schnorr, *RSA and Rabin Functions: Certain Parts are as Hard as the Whole*. SIAM Journal on Computing, v. 17, n. 2, pages 194-209, April 1988.

[Adl79] L.M. Adleman, *A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography*. Proceedings of the IEEE 20th Annual Symposium of Foundations of Computer Science, pages 55-60, 1979.

[Adl91] L.M. Adleman, *Factoring Numbers Using Singular Integers*. Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing, 1991, pages 64-71.

[AG99] M. Abadi, A.D. Gordon. *A calculus for cryptographic protocols: The spi calculus*. Information and Computation, 148(1), pages 1-70, January 1999.

[AHU74] A. Aho, J. Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Alk83] S.G. Akl, *Digital Signatures: A Tutorial Survey*. Computer, v. 16, n. 2, pages 15-24, February 1983.

[AN95] R.J. Anderson, R. Needham, *Robustness Principles for Public Key Protocols.* Advances in Cryptology-CRYPTO '95 Proceedings, Springer-Verlag, 1995.

[APF01] M. Alam, R. Prasad, J. Farserotu, *Quality of Service Among IP-Based Heterogeneous Networks*. IEEE Personal Communications, pages 18-24, December 2001.

[APR83] L.M. Adleman, C. Pomerance, R.S. Rumeley, *On Distinguishing Prime Numbers from Composite Numbers*. Annals of Mathematics, v. 117, n. 1, pages 173-206, 1983.

[Arm00] G. Armitage, *Quality of Service in IP Networks*. London: Pearson Higher Education, 2000.

[Aur97] T. Aura, *Strategies against replay attacks.* In 10th IEEE Computer Security Foundations Workshop, pages 59-68. IEEE Computer Society Press, 1997.

[BA02] A. Brand, H. Aghvami, *Multiple Access Protocols for Mobile Communications: GPRS, UMTS and Beyond.* John Wiley & Sons Ltd, ISBNs: 0-471-49877-7 (Hardback); 0-470-84622-4 (Electronic), 2002.

[BAN89] M. Burrows, M. Abadi, R. Needham, *A logic of authentication.* Proceedings of the Royal Society London, A426:233-271, 1989.

[BAN90] M. Burrows, M. Abadi, R. Needham, *A logic of authentication.* ACM Transactions on Computer Systems, 8(1):16-36, February 1990.

[BAN91] M. Burrows, M. Abadi, R. Needham, *The scope of a logic of authentication.* In J. Feigenabum et al., editors, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, volume 2, pages 119-126 AMS/ACM, 1991.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*. RFC 2475, Internet Engineering Task Force (IETF), December1998.

[BCK96] M. Bellare, R. Canetti, H. Krawczyk, *Keying hash functions for message authentication.* In N. Koblitz, editor, Advances in Cryptology - Crypto '96, pages 1-15, Springer-Verlag, Lecture Notes in Computer Science Volume 1109, 1996.

[BCS94] R. Braden, D. Clark, S. Shenker, *Integrated Services in the Internet Architecture: An Overview*. RFC 1633, Internet Engineering Task Force (IETF), Network Working Group, June 1994.

[BCY92] M. J. Beller, L.-F. Chang, Y. Yacobi, *Security for personal communication services: Public-key vs. private key approaches.* In 3rd IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'92), pages 26-31, IEEE Press, October 1992

[BHR99] S. Boeyen, T. Howes, P. Richard, *Internet X.509 Public Key Infrastructure LDAPv2 Schema.* RFC 2587, Internet Engineering Task Force (IETF), June 1999.

[BKP00] V. Boyko, P. MacKenzie, S. Patel. *Provably secure password authenticated key exchange using Diffie-Hellman.* In B. Preneel, editors, Advances in Cryptology - Eurocrypt 2000, pages 156-171, Springer-Verlag, Lecture Notes in Computer Science Volume 1807, 2000.

[BKW05] V. Bollapragada, M. Khalid, S. Wainner, *IPSec VPN Design*. Cisco Press, April 2005.

[BM00] C. Boyd, A. Mathuria, *Key establishment protocols for secure mobile communications: A critical survey.* Computer Communications, 23, pages 575-587, 2000.

[BM94] C. Boyd, W. Mao, *On a limitation of BAN logic.* In T. Helleseth, editor, Advances in Cryptology - Eurocrypt '93, pages 240-247. Springer-Verlag, Lecture Notes in Computer Science Volume 765, 1994.

[BM97] C. Boyd, A. Mathuria, *Systematic design of key establishment protocols based on one-way functions.* IEE Proceedings - Computers and Digital Techniques, 144(2), pages 93-99, 1997.

[BM99] S. Blake-Wilson, A. Menezes, *Authenticated Diffie-Hellman key agreement protocols.* In S. Tavares et al., editors, Selected Areas in Cryptography, 5th International Workshop, pages 339-361. Springer-Verlag, 1999.

[Bor00] M. S. Borella, *Methods and protocols for secure key negotiation using IKE.* IEEE Networks, 14(4), pages 18-29, July/August 2000.

[Boy93] C. Boyd, *Security architectures using formal methods.* IEEE Journal on Selected Areas in Communications, 11(5), pages 694-701, 1993.

[BR95] M. Bellare, P. Rogaway, *Optimal asymmetric encryption - how to encrypt with RSA.* In A. De Santis, Advances in Cryptology - Eurocrypt '94 pages 92-111, Springer-Verlag, Lecture Notes in Computer Science Volume 950, 1995

[Bri01] A. Bria, *Fourth-Generation Wireless Infrastructures: Scenarios and Research Challenges*. IEEE Personal Communications, Vol. 8, No. 6, pages 25-31, December 2001.

[Cam00] A. T. Campbell, *Design, Implementation, and Evaluation of Cellular IP*. IEEE Personal Communications, pages 42-49, August 2000.

[CB94] W. R. Cheswick, S. M. Bellovin, *Firewalls and Internet Security*. Reading, MA: Addison-Wesley, 1994.

[CL03] P. Calhoun, J. Loughney, *Diameter Base Protocol*, RFC 3588, Internet Engineering Task Force (IETF), September 2003.

[CM04] C. Cremers, S. Mauw, *Checking secrecy by means of partial order reduction.* In D. Amyor, A. Williams, editors, SAM 2004: Security Analysis and Modelling. 4th Workshop on SDL and MSC, pages 177-194, 2004.

[DH76] W. Diffie, M.E. Hellman, *New Directions in Cryptography*. IEEE Transactions on Information Theory, v. IT-22, n. 6, pages 644-654, November 1976.

[DH98] S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460, Internet Engineering Task Force (IETF), December 1998.

[Dix02] S. Dixit, *Wireless IP and Its Challenges for the Heterogeneous Environment*. International Journal of Wireless Personal Communications, August 2002.

[DO92] W. Diffie, P. C. van Oorschot, M.J. Winner, *Authentication and authenticated key exchange.* Designs, Codes and Cryptography, v 2, pages 107-125, 1992.

[DR02] J. Daemen, V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, ISBN 3540425802, 2002.

[Dro02] R. Droms, Ted Lemon, *DHCP Handbook.* Sams ,2nd Edition, ISBN: 0672323273, November 2002.

[EHS97] J. Ellsberger, D. Hogrefe, A. Sarma, *SDL:Formal Object-oriented Language for Communicating Systems.* Prentice Hall, 1997.

[FDR99] Formal Systems(Europe) Ltd. *FDR2 User Manual*, August 1999.

[FHS+04] X. Fu, D. Hogrefe, R. Soltwisch, S. Narayanan *QoS and Security in 4G Networks*. Proceedings of the 1st CIC/IEEE Global Mobile Congress (GMC 2004), Shanghai, China, pages 117-122, October 2004.

[Gam03] J. Garman, *Kerberos, The Definitve Guide*. O'Reilly Media, September 2003.

[Gam85] T. ElGamal, *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag, pages 10-18, 1985.

[Gam85a] T. ElGamal, *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. IEEE Transactions on Information Theory, v. IT-31, n. 4, pages 469-472, 1985.

[Geo04] M. Georgides, *Context transfer support for IP-based mobility management.* CCSR, UniS, Submission for the Research Excellence Awards Competition 2004, 2004.

[GM93] M.J.C. Gordon, T.F. Melham, *Introduction to HOL: A theorem proving environment for higher order logic.* 1993.

[GNY90] L. Gong, R. Needham, R. Yahalom, *Reasoning about belief in cryptographic protocols.* In IEEE Symposium on Security and Privacy, pages 234-248. IEEE Computer Society Press, 1990.

[Gol00] D. Gollmann, *On the verification of cryptographic protocols - a tale of two committees.* In S. Schneider and P.Ryan, editors, Workshop on Security Architectures and Information Flow, volume 32 of Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam, 2000.

[Gol01] D. Gollmann, *Authentication - myths and misconceptions.* Progress in Computer Science and Applied Logic, pages 203-225, 2001.

[Gol03] D. Gollmann, *Analysing security protocols.* In A. Abdallah, P. Ryan, S Schneider, editors, Formal Aspects of Security (FASec 2002), volume 2629 of Lecture Notes in Computer Science, pages 71-80, Springer, Berlin Heidelberg New York, 2003.

[Gol96] D. Gollmann, *What do we mean by entity authentication?* In IEEE Symposium on Security and Privacy, pages 46-54, IEEE Computer Society Press, 1996.

[Gon89] L. Gong, *Using one-way functions for authentication.* ACM Computer Communication Review, 19(5), pages 8-11, October 1989.

[Gon93] L. Gong, *Variations on the themes of message freshness and replay.* In 6th IEEE Computer Security Foundation Workshop, pages 131-136, IEEE Computer Society Press, June 1993.

[Gon95] L. Gong, *Collisionful keyed hash functions with selectable collisions.* Information Processing Letters, 55(3), pages 167-170, August 1995.

[Gop01] R. Gopal, *DOS Detection, Prevention and IDS System for Wireless Networks.* RSA Conference 2001, Amsterdam, October 2001.

[HC98] D. Harkins, D Carrel, *The Internet Key Exchange (IKE).* RFC 2409, Internet Engineering Task Force (IETF), November 1998.

[HCW01] C. Hoene, I. Carreras, A. Wolisz, *Voice over IP: Improving the Quality over Wireless LAN by Adopting a Booster Mechanism-An Experimental Approach.* Proc. ITCOM 2001, Denver, CO, August 2001.

[Hei98] Gunnar Heine, *GSM Networks: Protocols, Terminology, and Implementation.* Artech House, Franzis' Verlag, Boston London, 1998.

[HFP+99] R. Housley, W. Ford, W. Polk, D. Solo, *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile.* RFC 2459, January 1999.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985

[Hoa96] C. A. R. Hoare, *How did software get so reliable without proof?* In M.-C. Gaudel, J. Woodcock, editors, Formal Methods Europe 1996 (FME): Industrial Benefit and Advances in formal Methods, volume 1051 of Lecture Notes in Computer Science, pages 1-17, Springer, Berlin Heidelberg New York, 1996.

[Hog89] D. Hogrefe, *Estelle, LOTOS und SDL - Standard-Spezifkationssprachen* für verteilte Systeme. Springer Verlag, Berlin, Heidelberg, New York., 1989.

[HRM+03] Zhang Hong, He Rui, Yuan Man, Kan Zhigang and Qian Hualin, *A Novel Fast Authentication Method for Mobile Network Access*. International Conference for Young Computer Scientists (ICYCS), 2003.

[ITU92] International Telecommunication Union - Telecommunication Standardization Sector. *Recommendation Z.100 - CCITT Specification and Description Language (SDL).* ITU, Geneva, October 1992.

[ITU96] International Telecommunication Union - Telecommunication Standardization Sector. *Recommendation Z.120 - Message Sequence Charts (MSC).* ITU, Geneva, October 1996.

[ITU99] International Telecommunication Union - Telecommunication Standardization Sector. *Recommendation Z.100 - CCITT Specification and Description Language (SDL).* ITU, Geneva, November 1999.

[JPA04] D. Johnson, C. Perkins, J. Arkko, *Mobility Support in IPv6*. RFC 3775, Internet Engineering Task Force (IETF), Network Working Group, June 2004.

[Jue04] J. Jürjens, *Secure Systems Development with UML.* Springer-Verlag, Hardcover. ISBN: 3-540-00701-6, October 2004.

[KA98] S. Kent, R. Atkinson, *Security Architecture for the Internet Protocol.* RFC 2401, Internet Engineering Task Force (IETF), November 1998.

[KA98a] S. Kent, R. Atkinson, *IP Authentication Header.* RFC 2402, Internet Engineering Task Force (IETF), November 1998.

[KA98b] Kent, S., and R. Atkinson, *IP Encapsulating Security Payload*. RFC 2406, Internet Engineering Task Force (IETF), November 1998.

[Kal92] B. Kaliski, *The MD2 Message-Digest Algorithm*. RFC 1319, Internet Engineering Task Force (IETF), April 1992.

[KBC97] H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, Internet Engineering Task Force (IETF), February 1997.

[KKS01] V. Kumar, M. Korpi, S. Sengodan, *IP Telephony with H.323: Architectures for Unified Networks and Integrated Services*, New York: Wiley, 2001.

[Kne92] R. Kneuper, *Validation und Verification von Software durch symbolische Ausführungen.* In P. Liggesmeyer, H.M. Sneed, A. Spillner, Testen Analysieren und Verifizieren von Software, Informatik Aktuell, Springer Verlag, Berlin, 1992.

[Kor01] J. Korhonen, *Introduction to 3G Mobile Communications*. Norwood, MA: ArtechHouse, 2001.

[KPS02] C. Kaufman, R. Perlman, M. Speciner. *Network Security*, Pentrice Hall, 2002.

[Lai92] X. Lai, *On the Design and Security of Block Ciphers*. ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[LHY99] C.-H. Lee, M.-S. Hwang, W.-P. Yang, *Enhanced Privacy and Authentication for the Global System for Mobile Communications*, Wireless Networks, volume 5, issue 4, pages 231-243, 1999.

[LNP+05] J. Loughney, M. Nakhjiri, C. Perkins, R. Koodli, *Context Transfer Protocol (CxTP)*, Experimental RFC 4067, Internet Engineering Task Force (IETF), Network Working Group, July 2005.

[Low96] G. Lowe, *Braking and fixing the Needham-Schroeder public key protocol using FDR.* In Tools and Algorithms for the Construction and Analysis of Systems, pages 147-166. Springer-Verlag, 1996.

[Low97] G. Lowe, *Casper: A compiler for the analysis of security protocols.* 10th IEEE Computer Security Foundations Workshop, 1997.

[Low98] G. Lowe, *Casper: A compiler for the analysis of security protocols*. Journal of Computer Security, 6, 1998.

[Mah98] D. O'Mahony, *Universal Mobile Telecommunications Systems: The Fusion of Fixed and Mobile Networks*. IEEE Internet Computing, Vol. 2, No. 1, January/ February 1998, pages 49-56.

[Mas94] J.L. Massey, *SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm*. Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pages 1-17, 1994.

[Mau98] D. Maughan, *Internet Security Association and Key Management Potocol(ISAKMP)*. RFC 2408, Internet Engineering Task Force (IETF), November 1998.

[Mea96] C. Meadows, *The NRL Protocol Analyzer: An overview.* The Journal of Logic Programming, 26(2):113.131, 1996.

[Mea99] C. Meadows, *Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer.* In IEEE Symposium on Security and Privacy, pages 216-231. IEEE Computer Society Press, 1999.

[Mas95] J.L. Massey, *SAFER K-64: One Year Later*. K.U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995.

[MG98] C. Madson, R. Glenn, *The Use of HMAC-MD5 within ESP and AH.* RFC 2403, Internet Engineering Task Force (IETF), November 1998.

[MG98a] C. Madson, R. Glenn, *The Use of HMAC-SHA-1 within ESP and AH.* RFC 2404, Internet Engineering Task Force (IETF), November 1998.

[Mil89] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[MIS95] NIST, *Secure Hash Standard*. FIPS PUB 180-1, April 1995.

[MMS97] J. C. Mitchell, M. Mitchell, U. Stern, *Automated analysis of cryptographic protocols using Murϕ.* In IEEE Symposium on Security and Privacy, pages 141-151, IEEE Computer Society Press, 1997.

[NBS77] National Bureau of Standards, NBS FIPS PUB 46, *Data Encryption Standard*. National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[Nes90] D. M. Nessett, *A Critique of the Burrows, Abadi and Needham logic.* ACM Operating Systems Review, 24(2):35-38, April 1990.

[NIS94] National Institute of Standards and Technology, NIST FIPS PUB 186, *Digital Signature Standard*. U.S. Department of Commerce, May 1994.

[NS78] R. Needham, M. Schroeder, *Using encryption for authentication in large networks of computers*. Communications of the ACM, 21(12), 1978.

[Opn05] The OPNET Modeler, *http://www.opnet.com/products/modeler*. 2005.

[Orm98] H. Orman, *The OAKLEY Key Determination Protocol.* IETF RFC 2412, Internet Engineering Task Force (IETF), November 1998.

[Pau98] L. C. Paulson, *The inductive approach to verifying cryptographic protocols.* Journal of Computer Security, pages 85-128, 1998.

[Per02] C. Perkins, *Mobility Support in IPv4*. RFC 3344, Internet Engineering Task Force (IETF), August 2002.

[PK00] R. Perlman, C. Kaufman, *Key exchanging in IPsec: Analysis of IKE*. IEEE Internet Computing, 4(6):50-56, November-December 2000.

[PMK00] R. Prasad, W. Mohr, W. Konhauser, *Third Generation Mobile Communication Systems*. Norwood, MA: Artech House, 2000.

[Ram00] R. Ramjee, *IP Micro-Mobility Support Using HAWAII*. Internet Draft, I-D Status: Expired, Internet Engineering Task Force (IETF), July 2000.

[RC94] P. Rogaway and D. Coppersmith, *A Software-Oriented Encryption Algorithm.* Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pages 56-63, 1994.

[Rec04] J. Rech, *Wireless LANs.* Heise Verlag, ISBN: 3936931046, January 2004.

[RHK+02] C. Rensing, Hasan, M. Karsten, B. Stiller. *AAA: A Survey and a Policy-Based Architecture and Framework*. IEEE Network, pages 22-27, 2002.

[Riv90] R.L. Rivest, *The MD4 Message Digest Algorithm*. RFC 1186, Internet Engineering Task Force (IETF), October 1990.

[Riv91] R.L. Rivest, *The MD4 Message Digest Algorithm*. Advances in Cryptology-CRYPTO '90 Proceedings, Springer-Verlag, pages 303-311, 1991.

[Riv92] R.L. Rivest, *The MD4 Message-Digest Algorithm.* RFC 1320, Internet Engineering Task Force (IETF), April 1992.

[Riv92a] R.L. Rivest, *The MD5 Message-Digest Algorithm.* RFC 1321, Internet Engineering Task Force (IETF), April 1992.

[Riv92b] R.L. Rivest, *The RC4 Encryption Algorithm.* RSA Data Security, Inc., March 1992.

[Riv93] R.L. Rivest, *Dr. Ron Rivest on the Difficulty of Factoring*. Ciphertext: The RSA Newsletter, v. 1, n. 1, pages 6-8, Fall 1993.

[Riv95] R.L. Rivest, *The RC5 Encryption Algorithm*. Dr. Dobb's Journal, v. 20, n. 1, pages 146-148, January 95.

[Riv95a] R.L. Rivest, *The RC5 Encryption Algorithm*. K.U. Leuven Workshop on Cryptographic Algorithms, Springer-Verlag, 1995.

[Riv98] R.L. Rivest, *A Description of the RC2 Encryption Algorithm*. RFC 2268, Internet Engineering Task Force (IETF), March 1998.

[Rob94] M.J.B. Robshaw, *MD2, MD4, MD5, SHA, and Other Hash Functions*. Technical Report TR-101, Version 3.0, RSA Laboratories, July 1994.

[Ros82] A. W. Roscoe, *A mathematical theory of communicating processes*. D. Phil, Oxford University, 1982

[Ros97] A.W. Roscoe, *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[RS01] P.Y.A Ryan, S. A. Schneider, *Modeling and analysis of security protocols: the CSP Approach.* Addison-Wesley, 2001.

[RSA78] R. L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptostreams.* Communication of the ACM, 21(2), pages 120-126, February 1978.

[RVC01] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*. RFC 3031, Internet Engineering Task Force (IETF), January 2001.

[RW00] C. Rigney and S. Willens, *Remote Authentication Dial In User Service (RADIUS)*, RFC 2865, Internet Engineering Task Force (IETF), June 2000.

[SBG+03] L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller, *Efficient Authentication and Key Distribution in Wireless IP Networks*, IEEE Wireless Communications, Vol. 10 No. 6, December 2003.

[Sch04] G. Schäfer, *Security in Fixed and Wireless Networks.* John Wiley and Sons Ltd ISBN: 0470863706, January 2004.

[Sch91] B. Schneier, *One-Way Hash Functions*. Dr. Dobb's Journal, v. 16, n. 9, pages 148-151, September 1991.

[Sch94] B. Schneier, *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish).* Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pages 191-204, 1994.

[Sch94a] B. Schneier, *The Blowfish Encryption Algorithm*. Dr. Dobb's Journal, v. 19, n. 4, pages 38-40, April 1994.

[Sch99] S. A. Schneider, *Concurrent and Real-time Systems: the CSP Approach.* Addison-Wesley, 1999.

[SFN+04] R. Soltwisch, X. Fu, S. Narayanan, D Hogrefe, *A Method for Authentication and Key Exchange for Seamless Inter-Domain Handovers.* Proceedings of IEEE International Conference on Networks (ICON 2004), Singapore, November 2004.

[SH04] R. Soltwisch, D. Hogrefe, *A Survey on Network Security*. Technische Berichte des Instituts für Informatik an der Georg-August-Universität Göttingen, ISSN 1611-1044, Number IFI-TB-2004-04, October 2004.

[SR98] H. Schulzrinne, J Rosenberg, *A Comparison of SIP and H.323 for Internet Telephony*, 4pages, Proc NOSSDAV'98, July 1998.

[SR99] H. Schulzrinne, J Rosenberg, *The IETF Internet Telephony Architecture and Protocols*, IEEE Network, pages 18-23, May/June 1999.

[ST05] R. Soltwisch, F. Tegeler, *Review of CasperFDR Analysis of the IDKE Protocol*. Technical Report No. IFI-TB-2005-04, Institute for Informatics, University of Göttingen, Germany, ISSN 1611-1044, June 2005.

[STH05] R. Soltwisch, F. Tegeler, D. Hogrefe, *Formal Specification and Security Verification of the IDKE Protocol using FDR Model Checking*. IEEE International Conference on Networks (ICON), Kuala Lumpur, Malaysia, IEEE, November 2005.

[Tel01] Telelogic, *Tau 4.1 User's Manual.* Telelogic Malmö, Sweden, 2001.

[Tsu92] G. Tsudik, *Message Authentication with One-Way Hash Functions*. ACM Computer Communications Review, v. 22, n. 5, pages 29-38, 1992.

[Tun99] B. Tung, *Kerberos*, Addison-Wesley Professional, May 1999.

[Whe94] D.J. Wheeler, *A Bulk Data Encryption Algorithm.* Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pages 127-134, 1994.

[XHW94] S.B. Xu, D.K. He, X.M. Wang, *An Implementation of the GSM General Data Encryption Algorithm A5*. CHINACRYPT '94, Xidian, China, pages 287-291, November 1994.

[ZPS93] Y. Zheng, J. Pieprzyk, J. Seberry, *HAVAL-A One-Way Hashing Algorithm with Variable Length of Output*. Advances in Cryptology-AUSCRYPT '92 Proceedings, Springer-Verlag, pages 83-104, 1993.

Abbreviations

2G	Second Generation of mobile phones
3G	Third Generation of mobile phones (,e.g., GSM)
4G	Fourth Generation of mobile phones (also called beyond 3G)
AAA	Authentication Authorization and Accounting
AAAF	AAA-Server of Foreign Domain
AAAH	AAA-Server of Home Domain
AAAL	AAA-Server of the (local) access network
AES	Advanced Encryption Standard
AH	Authentication Header
AKA	Authentication and Key Agreement
AKE	Asymmetric Key Exchange
AP	Access Point
AR	Access Router
AVP	Attribute-Value-Pair
BS	Base Station
BSS	Base Station Subsystem
BU	Binding Update
CA	Certification Authority
cAR	current Access Router
CBC	Cipher-Block Chaining (Mode)
CBS	Cell Broadcast Service
CC	Client Challenge
CEFSM	Communicating Extended Finite State Machine
CHAP	Challenge Handshake Authentication Protocol
CK	Cipher Key
CN	Correspondent Node
CoA	Care-of Address

CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CSP	Communication Sequential Processes
CTP	Context Transfer Protocol, also called CxTP
CxTP	Context Transfer Protocol (former CTP)
DES	Data Encryption Standard
DH	Diffie-Hellman
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
DoS	Denial of Service
DSS	Digital Signature Standard
EAP	Extensible Authentication Protocol
EFSM	Exceeded Finite State Machine
ESP	Encapsulating Security Payload
FA	Foreign Agent
FDR	Failure Divergence Refinement
FIFO	first in first out
GSM	Global System for Mobile Communications
HA	Home Agent
HLR	Home Location Register
HMAC	Keyed-Hash Message Authentication Code
HOA	Home Agent Answer
HOR	Home Agent Request
HQ	high quality
HSS	Home Subscriber Server
ICV	Integrity Check Value
IDKE	Inter-Domain Key Exchange Protocol
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol

IPsec	Internet Protocol Security
IPv4	IP version 4
IPv6	IP version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISP	Internet Service Provider
IV	Initialization Vector
KDC	Key Distribution Centre
Ksms	Session Master Secret - A session-key
LAN	Local Area Network
LQ	low quality
MAC	Medium Access Control
MAC	Message Authentication Code
MAP	Mobile Application Part
MD2	Message Digest #2
MD3	Message Digest #3
MD4	Message Digest #4
MD5	Message Digest #5
ME	Mobile Equipment
MIP	Mobile IP
MIPv4	Mobile IP version 4
MIPv6	Mobile IP version 6
MIM	Man-in-the-Middle Attack
MN	Mobile Node
MPLS	Multi-Protocol Label Switching
MSC	Message Sequence Charts
MSC	Mobile Switching Centre
NAI	Network Access Identifier
NAP	Network Access Point
ND	Neighbor Discovery
OTP	One-Time Password

PFS	Perfect Forward Secrecy
PKCS	Public Key Cryptography Standards
PPP	Point-to-Point Protocol
PRF	Pseudo Random Function
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RFC	Request For Comments (denotes IETF standards)
RSA	Rivest Shamir Adleman - an asymmetric cipher
SA	Security Association
SAD	Security Association Database
SAP	Service Access Point
SDL	Specification and Description Language
SKE	Shared Key Exchange
SLA	Service Level Agreement
SPD	Security Policy Database
SPEKE	Simple Password-authenticated Exponential Key Exchange
SPI	Security Parameter Index
UMTS	Universal Mobile Telecommunications System
USIM	UMTS Subscriber Identity Module
VLR	Visited Location Register
VoIP	Voice over IP
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Networks, IEEE 802.11x
W-SKE	Wireless Shared Key Exchange Protocol

Appendix A: Common Security Protocol Syntax

The common syntax for protocol messages is defined by the following context free grammar, where (exp)* denotes 0 or more times exp, (exp)+ denotes 1 or more times exp, (exp)? denotes 0 or 1 time exp:

messages	:=(message (action)?)+
message	:= **label**'.' id '->' id':' **tuple**
label	:= id
id	:= non-empty sequence of alphanumeric characters, or '-', '_'
tuple	:= **atom** (',' **atom**)*
atom	:= cipher
	| clearterm
clearterm	:= id
	| **apply**
	| '(' (**tuple**)? ')'
cipher	:= { tuple }clearterm
apply	:= id'('**tuple**')'
action	:= arbitrary text

The identifiers (n.t. **id**) are protocol variables of a primitive types or functions, like one-way (hash) functions or functions which associate keys to principal names for instance. A primitive is proposed for encryption, with the usual notation using brackets. However, special identifier can also be used for encryption, when more details are needed.

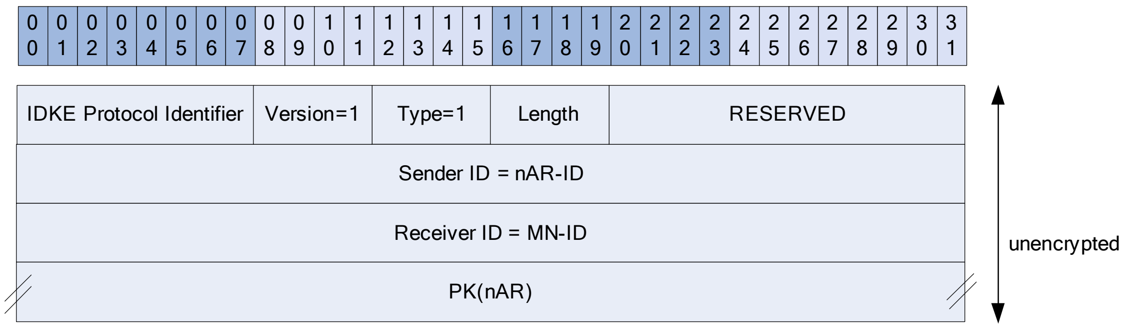
The identifiers can be declared before the protocol messages. The form of the type declaration of primitive or functional identifiers is free. In particular, in case of public key cryptography, a particular notation can be used to associate the public and private of a keypair.

Note that tuples can be written with or without parentheses. We assume by default that a tuple *a0, a1, ..., an* written without parentheses is left-associated.

Appendix B: Packet Formats of IDKE Messages

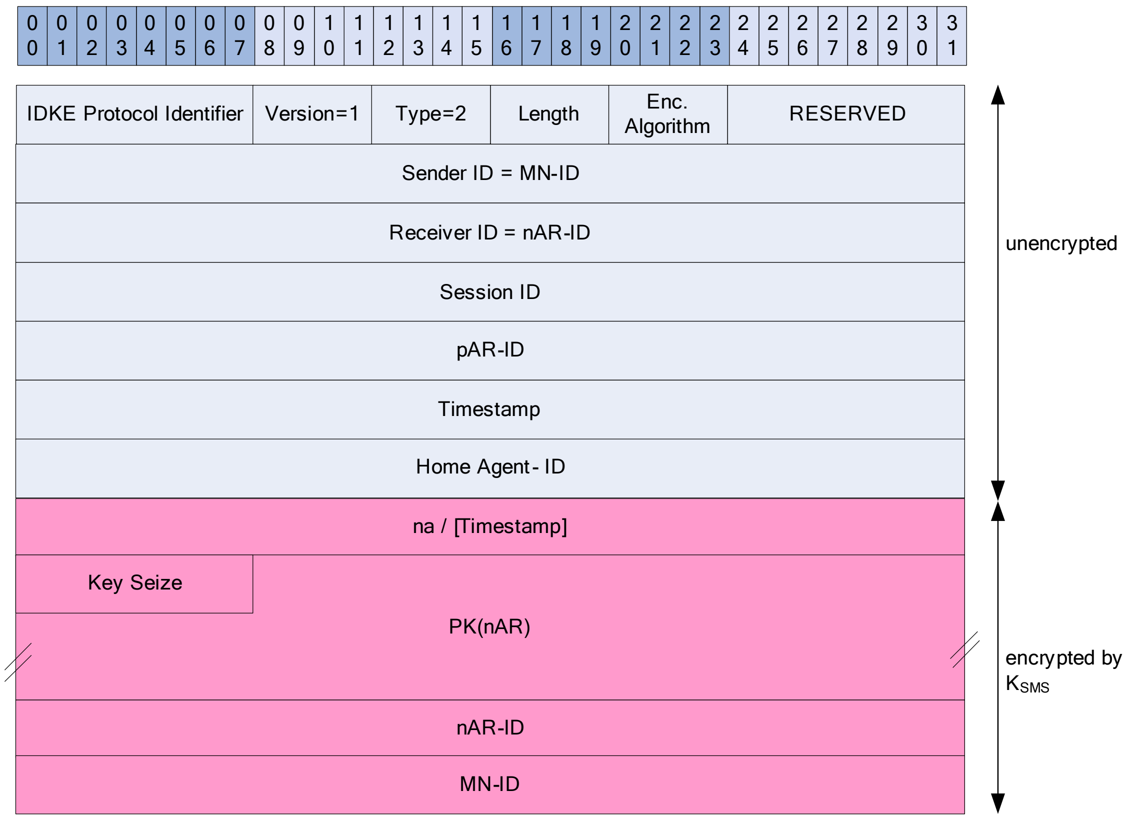
Packet formats of IDKE messages:

Message 1: from nAR to MN



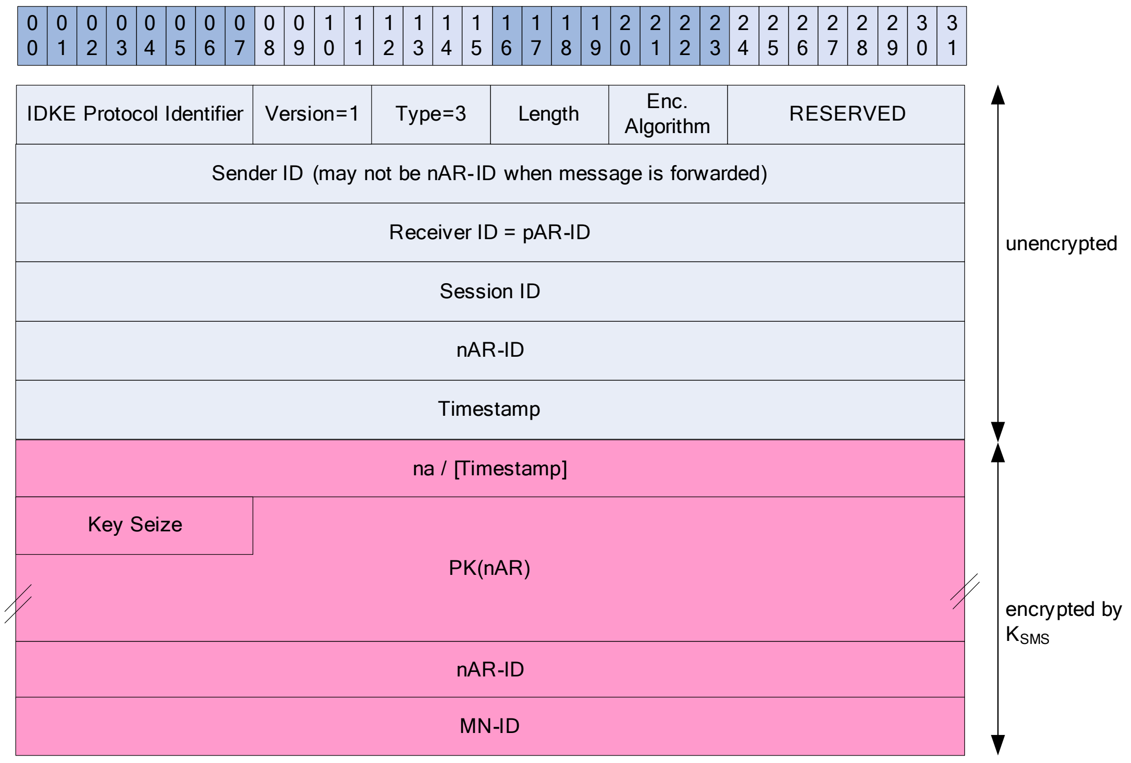
IDKE Message No. 1

Message 2: from MN to nAR



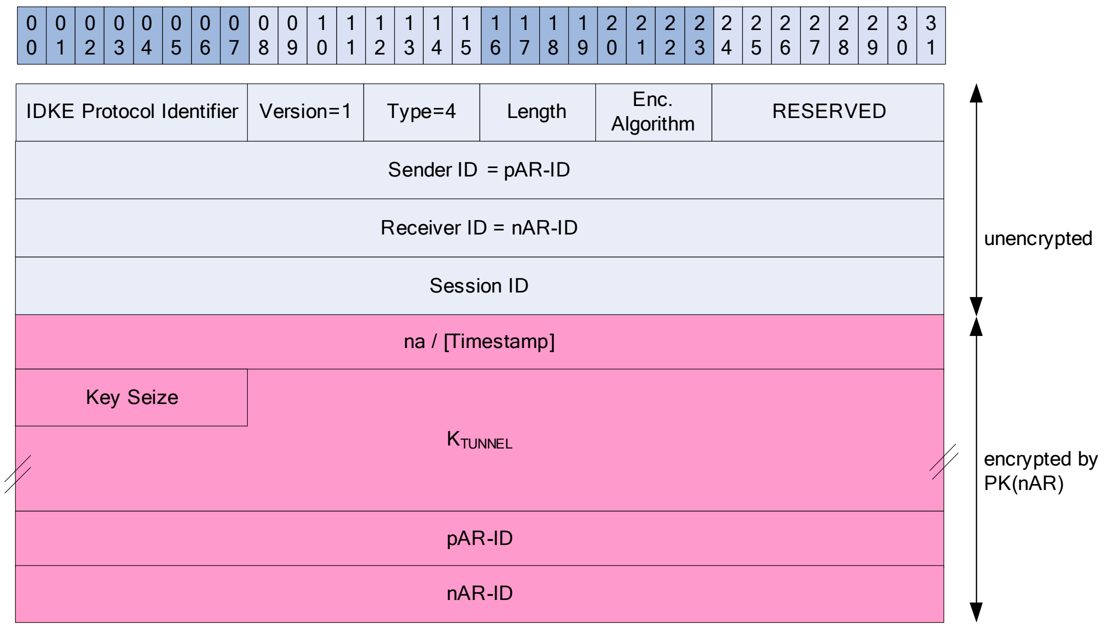
IDKE Message No. 2

Message 3: from nAR to pAR



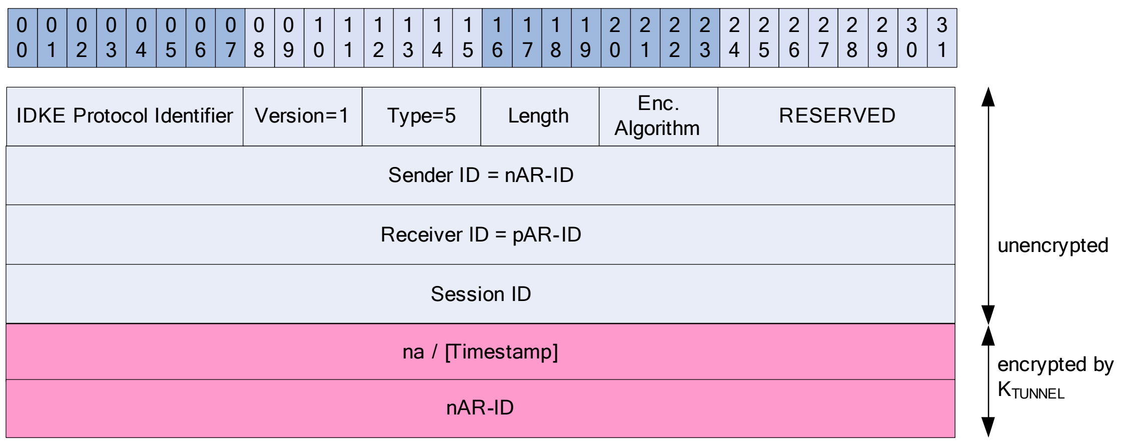
IDKE Message No. 3

Message 4: from pAR to nAR



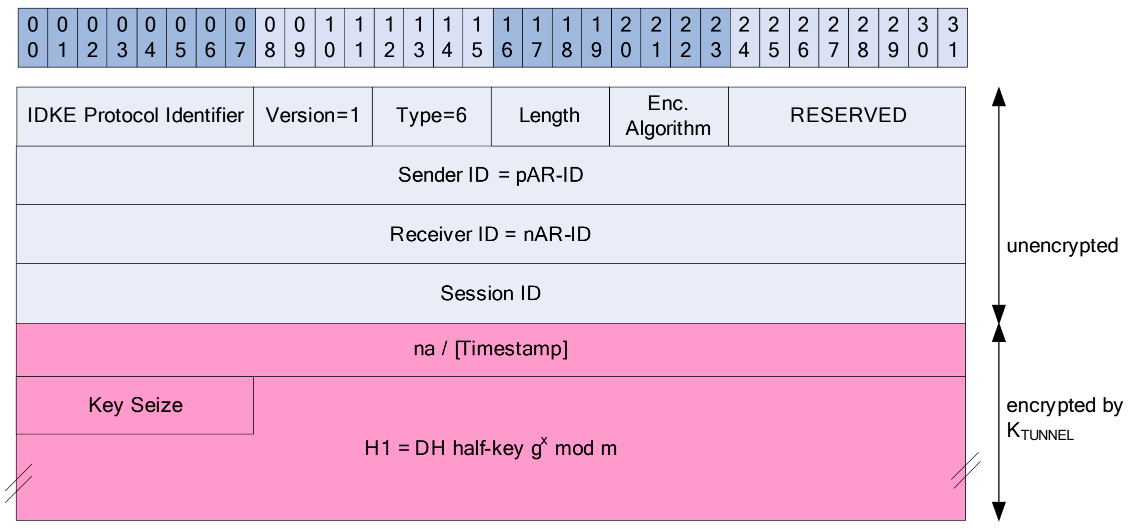
IDKE Message No. 4

Message 5: from nAR to pAR



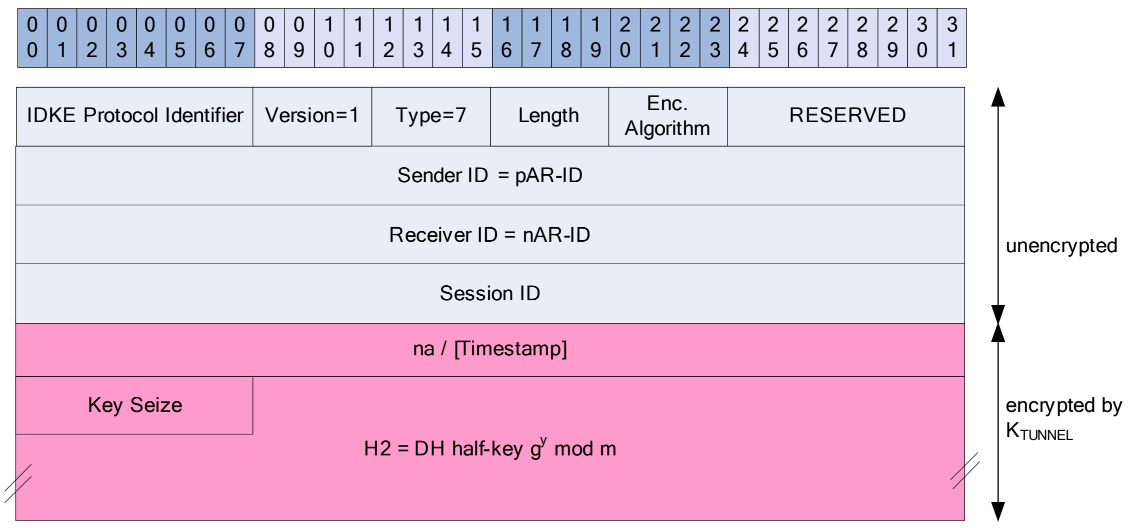
IDKE Message No. 5

Message 6: from pAR to nAR



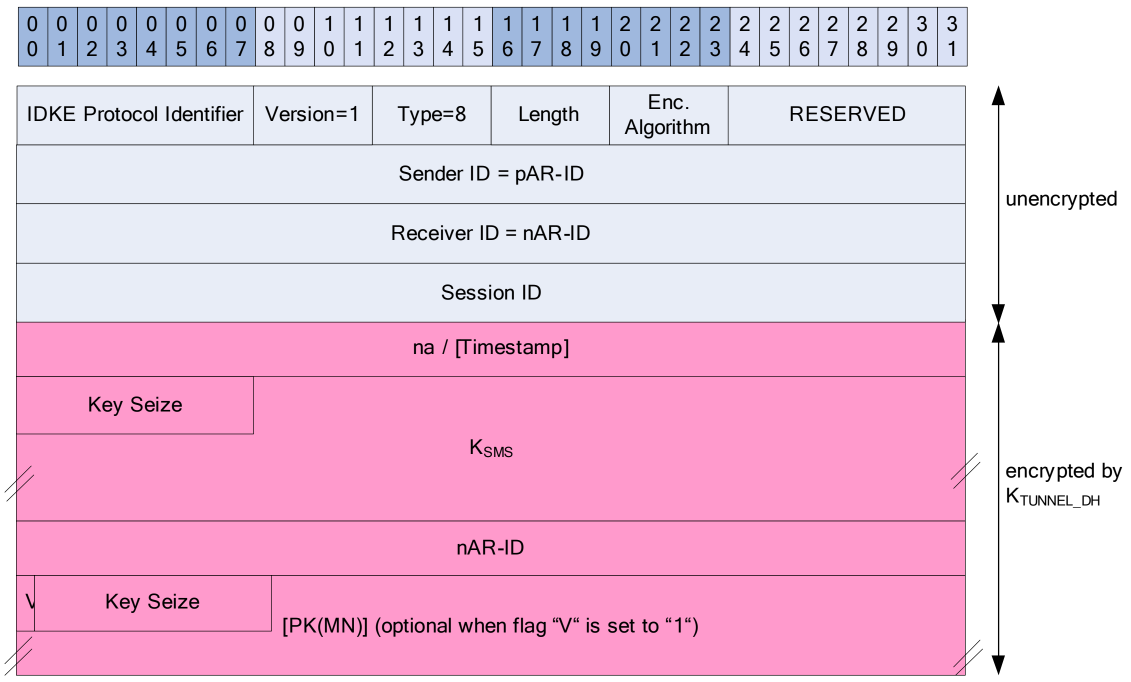
IDKE Message No. 6

Message 7: from nAR to pAR



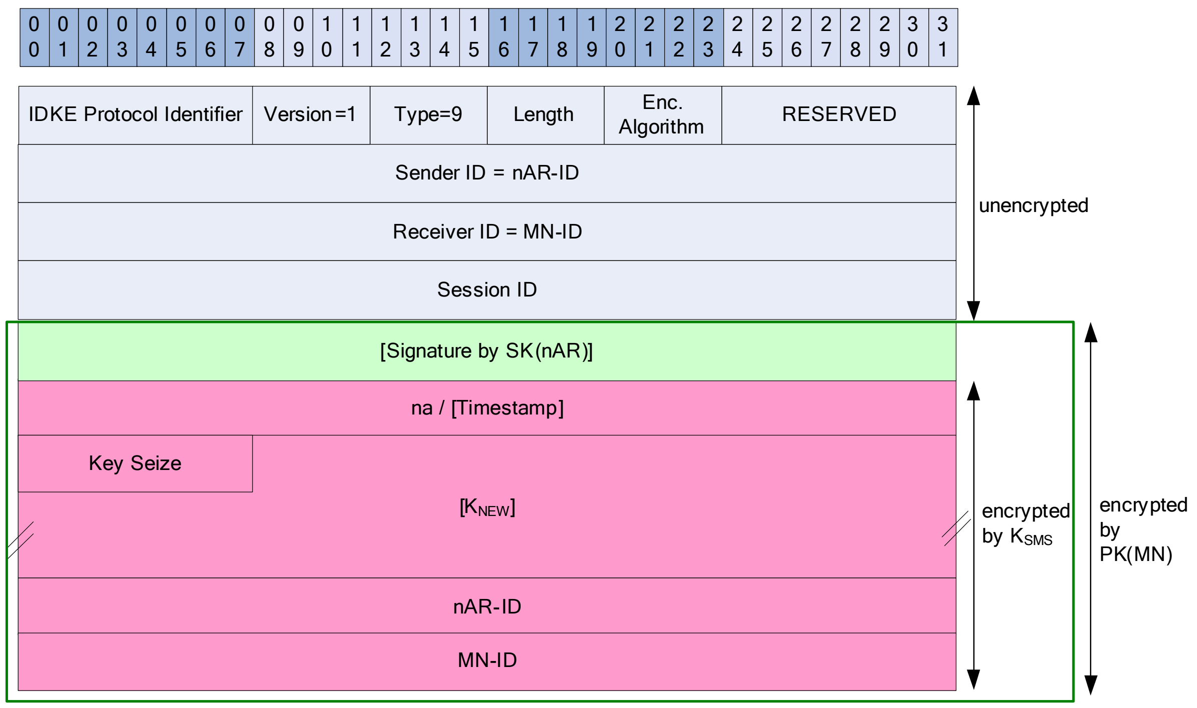
IDKE Message No. 7

Message 8: from nAR to pAR



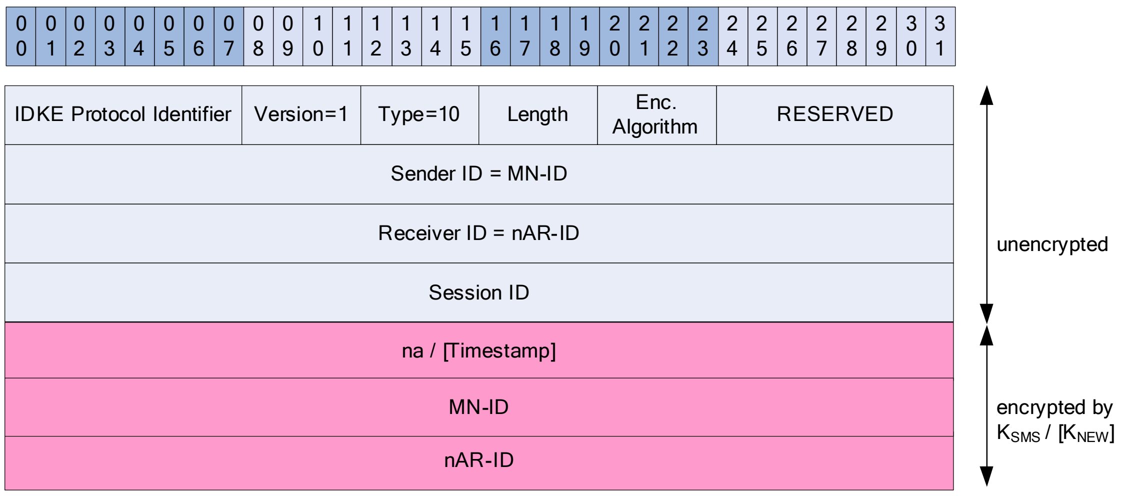
IDKE Message No. 8

Message 9: from nAR to MN



IDKE Message No. 9

Message 10: from MN to nAR



IDKE Message No. 10

Length: Message length in units of octets

RESERVED: Set to zero by the sender, ignored by the receiver

APPENDIX C: Casper, FDR & CSP

This appendix illustrates the Casper compiler output of the final IDKE protocol version as presented in Section 4.3.8. The CSP script files of additional versions are presented in a Technical Report that has been published in cooperation with Florian Tegeler [ST05].

The entire Casper script file, the FDR output and the corresponding CSP file are presented below:

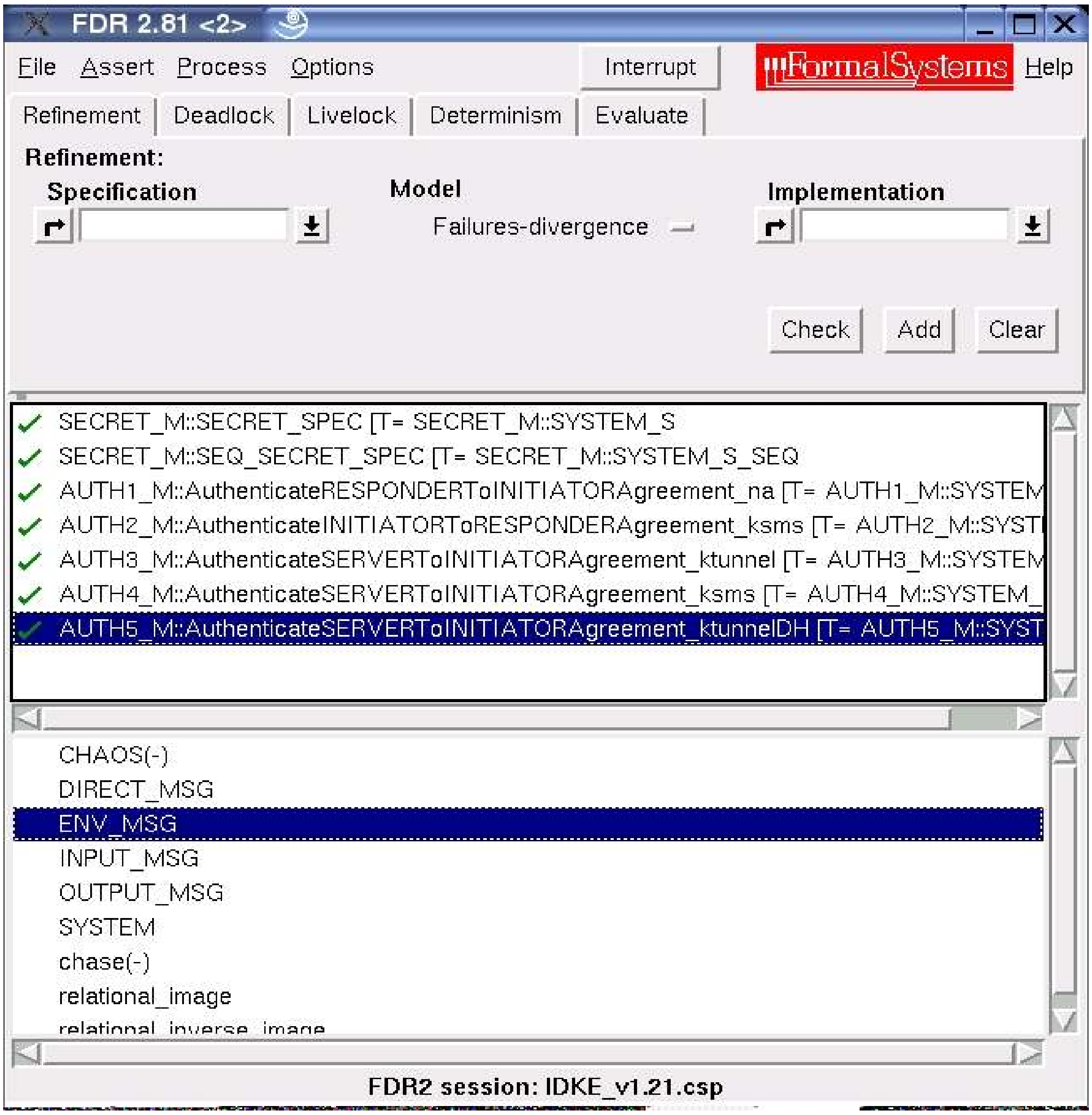
Casper script file:

```
#Free variables
datatype Field = Gen | Exp(Field,Num) unwinding 2
halfkeyA, halfkeyB, ktunnelDH : Field

M, N, P : Agent
pkn, pkp : PublicKey
skn, skp : SecretKey
ksms, ktunnel : SessionKey
na : Nonce
x, y : Num

InverseKeys = (pkn, skn), (ksms, ksms), (pkp, skp),(ktunnel, ktunnel),
(pkmallory, skmallory), (Exp, Exp), (Gen, Gen), (ktunnelDH, ktunnelDH)

#Protocol description
0.   -> N : M
1. N -> M : N, pkn
[N!=M]
2. M -> N : P, {pkn,N, M}{ksms} % token
[M!=N]
3. N -> P : N, token % {pkn, N, M}{ksms}
[N!=P]
4. P -> N : P, {ktunnel, P, N}{pkn}
[P!=N]
5. N -> P : {N}{ktunnel}
[N!=P]
-- SA Established - Agree on ktunnel -------
6. P -> N : {Exp(Gen,x) % halfkeyA}{ktunnel}
```

```
[P!=N]
< ktunnelDH := Exp(halfkeyA, y) >
7. N -> P : {Exp(Gen,y) % halfkeyB}{ktunnel}
[N!=P]
< ktunnelDH := Exp(halfkeyB, x) >
-- SA Established - Agree on ktunnelDH - PFS
8. P -> N : {ksms, N}{ktunnelDH}
[P!=N]
9. N -> M : {na, N, M}{ksms}
[N!=M]
10. M -> N : {na, M, N}{ksms}

#Equivalences
forall x, y : Num . Exp ( Exp(Gen,x), y ) = Exp( Exp(Gen,y), x )

#Processes
INITIATOR(N, pkn, skn, y, pkp, na)
RESPONDER(M, ksms, P)
SERVER(P,ksms, M, pkp, skp, pkn, ktunnel, x)

#Actual variables
MobileNode, NewAR, PrevAR, Mallory : Agent
PKN, PKP, PKMALLORY : PublicKey
SKN, SKP, SKMALLORY : SecretKey
KSMS, KTUNNEL : SessionKey
X, Y, Z : Num
Na, NInt : Nonce

InverseKeys = (KSMS, KSMS), (PKN, SKN), (PKP, SKP), (KTUNNEL, KTUNNEL),
(PKMALLORY, SKMALLORY)

#Specification
Secret(P, ktunnelDH, [N])
Secret(N, ktunnelDH, [P])
Secret(P, ktunnel, [N])
Secret(N, ktunnel, [P])
Secret(M, ksms, [N])
Secret(N, ksms, [M])
Secret(M, na, [N])
Secret(N, na, [M])
Agreement(M, N, [na])
Agreement(N, M, [ksms])
Agreement(P, N, [ktunnel])
Agreement(P, N, [ksms])
Agreement(P, N, [ktunnelDH])

#System
INITIATOR(NewAR, PKN, SKN, Y, PKP, Na)
RESPONDER(MobileNode, KSMS, PrevAR)
SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN, KTUNNEL, X)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {MobileNode, NewAR, Mallory, PKN, PKP, PrevAR,
PKMALLORY, SKMALLORY, Z, NInt}
```

FDR Output

Figure 52 shows the successful verification of the final IDKE protocol version. All security properties are verified as illustrated by green checkmarks.



```
✓ SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
✓ SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
✓ AUTH1_M::AuthenticateRESPONDERToINITIATORAgreement_na [T= AUTH1_M::SYSTEM
✓ AUTH2_M::AuthenticateINITIATORToRESPONDERAgreement_ksms [T= AUTH2_M::SYSTI
✓ AUTH3_M::AuthenticateSERVERToINITIATORAgreement_ktunnel [T= AUTH3_M::SYSTEM
✓ AUTH4_M::AuthenticateSERVERToINITIATORAgreement_ksms [T= AUTH4_M::SYSTEM_
✓ AUTH5_M::AuthenticateSERVERToINITIATORAgreement_ktunnelDH [T= AUTH5_M::SYST
```

```
CHAOS(-)
DIRECT_MSG
ENV_MSG
INPUT_MSG
OUTPUT_MSG
SYSTEM
chase(-)
relational_image
relational_inverse_image
```

FDR2 session: IDKE_v1.21.csp

Figure 52: FDR Screenshot – Output for the Final Version

CSP file

```
-- ***********************************************************************
-- *                                Types                                *
-- ***********************************************************************


-- Main datatype, representing all possible messages


datatype Encryption =
  MobileNode | NewAR | PrevAR | Mallory | PKN | PKP | PKMALLORY | SKN | SKP
|
  SKMALLORY | KSMS | KTUNNEL | X | Y | Z | Na | NInt | Garbage | Gen__ |
  Exp__.(Field, Num) | Sq.Seq(Encryption) |
  Encrypt.(ALL_KEYS,Seq(Encryption)) | Hash.(HashFunction, Seq(Encryption))
|
  Xor.(Encryption, Encryption)

-- All keys and hashfunctions in the system

ALL_KEYS = Union({SessionKey, Field, PublicKey})

ASYMMETRIC_KEYS = {k_, inverse(k_) | k_ <- ALL_KEYS, k_!=inverse(k_)}
HashFunction = {}

-- All atoms in the system

ATOM = {MobileNode, NewAR, PrevAR, Mallory, PKN, PKP, PKMALLORY, SKN, SKP,
         SKMALLORY, KSMS, KTUNNEL, X, Y, Z, Na, NInt, Garbage}

-- Some standard functions

encrypt(m_,k_) = Encrypt.(k_,m_)
decrypt(Encrypt.(k1_,m_),k_) = if k_ == inverse(k1_) then m_ else Garbage
decrypt(_,_) = Garbage
decryptable(Encrypt.(k1_,m_),k_) = k_ == inverse(k1_)
decryptable(_,_) = false
nth(ms_,n_) = if n_ == 1 then head(ms_) else nth(tail(ms_), n_ - 1)

nthts(ms_,n_) = if n_ == 1 then head(ms_) else nthts(tail(ms_), n_ - 1)
```

```
-- add Garbage to a set that contains and encryption,
-- hash function application of Vernam encryption

addGarbage_(S_) =
   if S_=={} then {Garbage}
   else Union({S_, {Garbage | Encrypt._ <- S_},
               {Garbage | Hash._ <- S_},
               {Garbage | Xor._ <- S_}})

-- Definitions of user supplied functions

Gen = Gen__

Exp(arg_1_, arg_2_) = Exp__.(arg_1_, arg_2_)

-- Inverses of functions

inverse(KSMS) = KSMS
inverse(PKN) = SKN
inverse(PKP) = SKP
inverse(KTUNNEL) = KTUNNEL
inverse(PKMALLORY) = SKMALLORY
inverse(SKN) = PKN
inverse(SKP) = PKP
inverse(SKMALLORY) = PKMALLORY
inverse(Exp__.arg_) = Exp__.arg_
inverse(Gen__) = Gen__

-- Types in system

Agent = {MobileNode, NewAR, PrevAR, Mallory}
PublicKey = {PKN, PKP, PKMALLORY}
SecretKey = {SKN, SKP, SKMALLORY}
SessionKey = {KSMS, KTUNNEL}
Num = {X, Y, Z}
Nonce = {Na, NInt}
```

```
Field__(n_) =
  if n_==0 then {Gen}
  else
    Union({
      {Gen},
      {Exp(arg_1, arg_2) | arg_1 <- Field__(n_-1), arg_2 <- Num}
    })
Field = Field__(2)

AllForegrounds = {}

-- ***********************************************************************
-- *                              Messages                               *
-- ***********************************************************************

-- Message labels

datatype Labels =
  Msg1 | Msg2 | Msg3 | Msg4 | Msg5 | Msg6 | Msg7 | Msg8 | Msg9 | Msg10 |
  Env0

MSG_BODY     =     {ALGEBRA_M::applyRenaming(m_)     |     (_,m_,_,_)     <-
SYSTEM_M::INT_MSG_INFO}

-- Type of principals

ALL_PRINCIPALS = Agent

HONEST = diff(ALL_PRINCIPALS, {Mallory})

-- Channel declarations

INPUT_MSG = SYSTEM_M::INPUT_MSG
OUTPUT_MSG = SYSTEM_M::OUTPUT_MSG
DIRECT_MSG = SYSTEM_M::DIRECT_MSG
ENV_MSG = SYSTEM_M::ENV_MSG

channel receive: ALL_PRINCIPALS.ALL_PRINCIPALS.INPUT_MSG
channel send: ALL_PRINCIPALS.ALL_PRINCIPALS.OUTPUT_MSG
channel env : ALL_PRINCIPALS.ENV_MSG
channel error
channel start, close : HONEST.HONEST_ROLE

channel leak : addGarbage_(ALL_SECRETS_DI)
-- Roles of agents

datatype ROLE = SPY_ | INITIATOR_role | RESPONDER_role | SERVER_role

HONEST_ROLE = diff (ROLE, {SPY_})

-- Secrets in the protocol

ALL_SECRETS_0 =
  Union({
    SessionKey,
    SessionKey,
    SessionKey,
    SessionKey,
    Nonce,
    Nonce
  })
ALL_SECRETS = addGarbage_(ALGEBRA_M::applyRenamingToSet(ALL_SECRETS_0))

  ALL_SECRETS_DI = ALL_SECRETS

-- Define type of signals, and declare signal channel

datatype Signal =
  Claim_Secret.ALL_PRINCIPALS.ALL_SECRETS.Set(ALL_PRINCIPALS) |
  Running1.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.Nonce |
  Commit1.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.Nonce |
  RunCom1.ALL_PRINCIPALS.ALL_PRINCIPALS.Nonce.Nonce |
  Running2.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  Commit2.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  RunCom2.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey.SessionKey |
  Running3.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  Commit3.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  RunCom3.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey.SessionKey |
  Running4.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  Commit4.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey |
  RunCom4.ALL_PRINCIPALS.ALL_PRINCIPALS.SessionKey.SessionKey |
  Running5.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.Field |
  Commit5.HONEST_ROLE.ALL_PRINCIPALS.ALL_PRINCIPALS.Field |
  RunCom5.ALL_PRINCIPALS.ALL_PRINCIPALS.Field.Field

channel signal : Signal

Fact_1 =
  Union({
    {Garbage},
    Field,
    Agent,
    PublicKey,
    SecretKey,
    SessionKey,
    Nonce,
    Num,
    {Encrypt.(ksms, <pkn, N, M>) |
       M <- Agent, N <- Agent, ksms <- SessionKey, pkn <- PublicKey},
    {Encrypt.(pkn, <ktunnel, P, N>) |
       N <- Agent, P <- Agent, ktunnel <- SessionKey, pkn <- PublicKey},
    {Encrypt.(ktunnel, <N>) |
       N <- Agent, ktunnel <- SessionKey},
    {Encrypt.(ktunnel, <halfkeyA>) |
       ktunnel <- SessionKey, halfkeyA <- addGarbage_(Field)},
    {Sq.<Gen, x> |
       x <- Num},
    {Encrypt.(ktunnel, <Exp(Gen, x)>) |
       ktunnel <- SessionKey, x <- Num},
    {Encrypt.(ktunnel, <halfkeyB>) |
       ktunnel <- SessionKey, halfkeyB <- addGarbage_(Field)},
    {Sq.<Gen, y> |
       y <- Num},
    {Encrypt.(ktunnel, <Exp(Gen, y)>) |
       ktunnel <- SessionKey, y <- Num},
    {Encrypt.(ktunnelDH, <ksms, N>) |
       N <- Agent, ksms <- SessionKey, ktunnelDH <- Field},
    {Encrypt.(Gen, <ksms, N>) |
       N <- Agent, ksms <- SessionKey},
    {Encrypt.(Exp(Gen, Num_1), <ksms, N>) |
       N <- Agent, Num_1 <- Num, ksms <- SessionKey},
    {Encrypt.(Exp(Exp(Gen, Num_2), Num_1), <ksms, N>) |
       N <- Agent, Num_1 <- Num, Num_2 <- Num, ksms <- SessionKey},
    {Encrypt.(Exp(Exp(Gen, Num_1), Num_2), <ksms, N>) |
       N <- Agent, Num_1 <- Num, Num_2 <- Num, ksms <- SessionKey},
    {Encrypt.(ksms, <na, N, M>) |
       M <- Agent, N <- Agent, ksms <- SessionKey, na <- Nonce},
    {Encrypt.(ksms, <na, M, N>) |
       M <- Agent, N <- Agent, ksms <- SessionKey, na <- Nonce},
    {Sq.<Gen, Num_1> |
       Num_1 <- Num},
    {Sq.<Exp(Gen, Num_1), Num_2> |
       Num_1 <- Num, Num_2 <- Num},
    {Sq.<Exp(Gen, Num_2), Num_1> |
       Num_1 <- Num, Num_2 <- Num},
    {Sq.<Gen, Num_2> |
       Num_2 <- Num}
  })

external relational_inverse_image
external relational_image
transparent chase

-- ***********************************************************************
-- *                           Honest Agents                             *
-- ***********************************************************************

module SYSTEM_M

  -- Environmental messages

  ENV_INT_MSG0 =
    {(Env0, M, <>) |
       M <- Agent}

  ENV_MSG0 = {ALGEBRA_M::rmb(m_) | m_ <- ENV_INT_MSG0}

  ENV_INT_MSG = ENV_INT_MSG0

  -- information about messages sent and received by agents, including
  -- extras fields for both agents

  INT_MSG_INFO1 =
    {(Msg1, Sq.<N, pkn>, <>, <>) |
       N <- Agent, pkn <- PublicKey}
  INT_MSG_INFO2 =
    {(Msg2, Sq.<P, token>, <>, <>) |
       P <- Agent,
       token <- addGarbage_({Encrypt.(ksms, <pkn, N, M>) | M <- Agent, N <-
Agent, ksms <- SessionKey, pkn <- PublicKey})}
  INT_MSG_INFO3 =
    {(Msg3, Sq.<N, token>, <>, <>) |
       N <- Agent,
       token <- addGarbage_({Encrypt.(ksms, <pkn, N, M>) | M <- Agent, N <-
Agent, ksms <- SessionKey, pkn <- PublicKey})}
  INT_MSG_INFO4 =
    {(Msg4, Sq.<P, Encrypt.(pkn, <ktunnel, P, N>)>, <>, <>) |
       N <- Agent, P <- Agent, ktunnel <- SessionKey, pkn <- PublicKey}
  INT_MSG_INFO5 =
    {(Msg5, Encrypt.(ktunnel, <N>), <>, <>) |
       N <- Agent, ktunnel <- SessionKey}
  INT_MSG_INFO6 =
    {(Msg6, Encrypt.(ktunnel, <halfkeyA>), <>, <>) |
       ktunnel <- SessionKey, halfkeyA <- addGarbage_(Field)}
  INT_MSG_INFO7 =
    {(Msg7, Encrypt.(ktunnel, <halfkeyB>), <>, <>) |
       ktunnel <- SessionKey, halfkeyB <- addGarbage_(Field)}
  INT_MSG_INFO8 =
    {(Msg8, Encrypt.(ktunnelDH, <ksms, N>), <ktunnel, ksms, ktunnelDH>, <>)
|
       ktunnel <- SessionKey, ksms <- SessionKey, ktunnelDH <- Field,
       N <- Agent}
  INT_MSG_INFO9 =
    {(Msg9, Encrypt.(ksms, <na, N, M>), <ksms>, <>) |
       ksms <- SessionKey, M <- Agent, N <- Agent, na <- Nonce}
  INT_MSG_INFO10 =
    {(Msg10, Encrypt.(ksms, <na, M, N>), <na, ksms>, <na, P, ktunnel, ksms,
ktunnelDH>) |
       na <- Nonce, ksms <- SessionKey, P <- Agent, ktunnel <- SessionKey,
       ktunnelDH <- Field, M <- Agent, N <- Agent}

  -- types of messages sent and received by agents, as they are
  -- considered by those agents

  input_proj((l_,m_,se_,re_)) = (l_,m_,re_)
  rmb_input_proj((l_,m_,se_,re_)) = ALGEBRA_M::rmb((l_,m_,re_))
  output_proj((l_,m_,se_,re_)) = (l_,m_,se_)

  INPUT_INT_MSG1 = { input_proj(mt_) | mt_ <- INT_MSG_INFO1 }
  INPUT_INT_MSG2 = { input_proj(mt_) | mt_ <- INT_MSG_INFO2 }
  INPUT_INT_MSG3 = { input_proj(mt_) | mt_ <- INT_MSG_INFO3 }
  INPUT_INT_MSG4 = { input_proj(mt_) | mt_ <- INT_MSG_INFO4 }
  INPUT_INT_MSG5 = { input_proj(mt_) | mt_ <- INT_MSG_INFO5 }
  INPUT_INT_MSG6 = { input_proj(mt_) | mt_ <- INT_MSG_INFO6 }
  INPUT_INT_MSG7 = { input_proj(mt_) | mt_ <- INT_MSG_INFO7 }
  INPUT_INT_MSG8 = { input_proj(mt_) | mt_ <- INT_MSG_INFO8 }
  INPUT_INT_MSG9 = { input_proj(mt_) | mt_ <- INT_MSG_INFO9 }
  INPUT_INT_MSG10 = { input_proj(mt_) | mt_ <- INT_MSG_INFO10 }

  INPUT_INT_MSG =

    Union({
      INPUT_INT_MSG1,
      INPUT_INT_MSG2,
      INPUT_INT_MSG3,
      INPUT_INT_MSG4,
      INPUT_INT_MSG5,
      INPUT_INT_MSG6,
      INPUT_INT_MSG7,
      INPUT_INT_MSG8,
      INPUT_INT_MSG9,
      INPUT_INT_MSG10
    })

  OUTPUT_INT_MSG1 = { output_proj(mt_) | mt_ <- INT_MSG_INFO1 }
  OUTPUT_INT_MSG2 = { output_proj(mt_) | mt_ <- INT_MSG_INFO2 }
  OUTPUT_INT_MSG3 = { output_proj(mt_) | mt_ <- INT_MSG_INFO3 }
  OUTPUT_INT_MSG4 = { output_proj(mt_) | mt_ <- INT_MSG_INFO4 }
  OUTPUT_INT_MSG5 = { output_proj(mt_) | mt_ <- INT_MSG_INFO5 }
  OUTPUT_INT_MSG6 = { output_proj(mt_) | mt_ <- INT_MSG_INFO6 }
  OUTPUT_INT_MSG7 = { output_proj(mt_) | mt_ <- INT_MSG_INFO7 }
  OUTPUT_INT_MSG8 = { output_proj(mt_) | mt_ <- INT_MSG_INFO8 }
  OUTPUT_INT_MSG9 = { output_proj(mt_) | mt_ <- INT_MSG_INFO9 }
  OUTPUT_INT_MSG10 = { output_proj(mt_) | mt_ <- INT_MSG_INFO10 }

  OUTPUT_INT_MSG =

    Union({
      OUTPUT_INT_MSG1,
      OUTPUT_INT_MSG2,
      OUTPUT_INT_MSG3,
      OUTPUT_INT_MSG4,
      OUTPUT_INT_MSG5,
      OUTPUT_INT_MSG6,
      OUTPUT_INT_MSG7,
      OUTPUT_INT_MSG8,
      OUTPUT_INT_MSG9,
      OUTPUT_INT_MSG10
    })

  -- INITIATOR

  INITIATOR_0(N, pkn, skn, y, pkp, na) =
    [] M : Agent @ env_I.N.(Env0, M,<>) ->
    output.N.M.(Msg1, Sq.<N, pkn>,<>) ->
    [] P : Agent @
```

```
    [] token : addGarbage_({Encrypt.(ksms, <pkn, N, M>) | M <- Agent, N <-
Agent, ksms <- SessionKey, pkn <- PublicKey}) @
      M!=N & input.M.N.(Msg2, Sq.<P, token>,<>) ->
    output.N.P.(Msg3, Sq.<N, token>,<>) ->
    [] ktunnel : SessionKey @ P!=N &
      input.P.N.(Msg4, Sq.<P, Encrypt.(inverse(skn), <ktunnel, P, N>)>,<>)
->
    output.N.P.(Msg5, Encrypt.(ktunnel, <N>),<>) ->
    [] halfkeyA : addGarbage_(Field) @ P!=N &
      true and member((Msg6, Encrypt.(inverse(ktunnel), <halfkeyA>),<>),
INPUT_INT_MSG6) &
      input.P.N.(Msg6, Encrypt.(inverse(ktunnel), <halfkeyA>),<>) ->
    INITIATOR_0'(N, pkn, skn, y, pkp, na, M, P, token, ktunnel, halfkeyA,
Exp(halfkeyA,y))


INITIATOR_0'(N, pkn, skn, y, pkp, na, M, P, token, ktunnel, halfkeyA,
ktunnelDH) =
    true and member((Msg7, Encrypt.(ktunnel, <Exp(Gen, y)>),<>),
OUTPUT_INT_MSG7) &
    output.N.P.(Msg7, Encrypt.(ktunnel, <Exp(Gen, y)>),<>) ->
    [] ksms : SessionKey @ P!=N &
      true and member((Msg8, Encrypt.(inverse(ktunnelDH), <ksms, N>),<>),
INPUT_INT_MSG8) &
      input.P.N.(Msg8, Encrypt.(inverse(ktunnelDH), <ksms, N>),<>) ->
    output.N.M.(Msg9, Encrypt.(ksms, <na, N, M>),<ksms>) ->
    input.M.N.(Msg10, Encrypt.(inverse(ksms), <na, M, N>),<na, P, ktunnel,
ksms, ktunnelDH>) ->
    SKIP


  INITIATOR_1(N, pkn, skn, y, pkp, na) = INITIATOR_0(N, pkn, skn, y, pkp,
na)


  INITIATOR(N, pkn, skn, y, pkp, na) =
    INITIATOR_1(N, pkn, skn, y, pkp, na)
      [[input.M.N.(l_,m_,re_) <- receive.M.N.ALGEBRA_M::rmb((l_,m_,re_)) |
          M <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO2]]
      [[input.P.N.(l_,m_,re_) <- receive.P.N.ALGEBRA_M::rmb((l_,m_,re_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO4]]
      [[input.P.N.(l_,m_,re_) <- receive.P.N.ALGEBRA_M::rmb((l_,m_,re_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO6]]
      [[input.P.N.(l_,m_,re_) <- receive.P.N.ALGEBRA_M::rmb((l_,m_,re_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO8]]
      [[input.M.N.(l_,m_,re_) <- receive.M.N.ALGEBRA_M::rmb((l_,m_,re_)) |
          M <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO10]]
      [[output.N.M.(l_,m_,se_) <- send.N.M.ALGEBRA_M::rmb((l_,m_,se_)) |
          M <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO1]]
      [[output.N.P.(l_,m_,se_) <- send.N.P.ALGEBRA_M::rmb((l_,m_,se_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO3]]
      [[output.N.P.(l_,m_,se_) <- send.N.P.ALGEBRA_M::rmb((l_,m_,se_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO5]]
      [[output.N.P.(l_,m_,se_) <- send.N.P.ALGEBRA_M::rmb((l_,m_,se_)) |
          P <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO7]]
      [[output.N.M.(l_,m_,se_) <- send.N.M.ALGEBRA_M::rmb((l_,m_,se_)) |
          M <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO9]]
      [[env_I.N.m_ <- env.N.ALGEBRA_M::rmb(m_) |
          m_ <- ENV_INT_MSG0]]

  -- RESPONDER

  RESPONDER_0(M, ksms, P) =
    [] N : Agent @ [] pkn : PublicKey @ N!=M &
      input.N.M.(Msg1, Sq.<N, pkn>,<>) ->
    output.M.N.(Msg2, Sq.<P, Encrypt.(ksms, <pkn, N, M>)>,<>) ->
    [] na : Nonce @ N!=M &
      input.N.M.(Msg9, Encrypt.(inverse(ksms), <na, N, M>),<>) ->
    output.M.N.(Msg10, Encrypt.(ksms, <na, M, N>),<na, ksms>) ->
    SKIP

  RESPONDER_1(M, ksms, P) = RESPONDER_0(M, ksms, P)

  RESPONDER(M, ksms, P) =
    RESPONDER_1(M, ksms, P)
      [[input.N.M.(l_,m_,re_) <- receive.N.M.ALGEBRA_M::rmb((l_,m_,re_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO1]]
      [[input.N.M.(l_,m_,re_) <- receive.N.M.ALGEBRA_M::rmb((l_,m_,re_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO9]]
      [[output.M.N.(l_,m_,se_) <- send.M.N.ALGEBRA_M::rmb((l_,m_,se_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO2]]
      [[output.M.N.(l_,m_,se_) <- send.M.N.ALGEBRA_M::rmb((l_,m_,se_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO10]]

  -- SERVER

  SERVER_0(P, ksms, M, pkp, skp, pkn, ktunnel, x) =
    [] N : Agent @ N!=P &
      input.N.P.(Msg3, Sq.<N, Encrypt.(inverse(ksms), <pkn, N, M>)>,<>) ->
    output.P.N.(Msg4, Sq.<P, Encrypt.(pkn, <ktunnel, P, N>)>,<>) ->
    N!=P & input.N.P.(Msg5, Encrypt.(inverse(ktunnel), <N>),<>) ->
    true and member((Msg6, Encrypt.(ktunnel, <Exp(Gen, x)>),<>), OUTPUT_INT_MSG6) &
    output.P.N.(Msg6, Encrypt.(ktunnel, <Exp(Gen, x)>),<>) ->
    [] halfkeyB : addGarbage_(Field) @ N!=P &
      true and member((Msg7, Encrypt.(inverse(ktunnel), <halfkeyB>),<>), INPUT_INT_MSG7) &
      input.N.P.(Msg7, Encrypt.(inverse(ktunnel), <halfkeyB>),<>) ->
    SERVER_0'(P, ksms, M, pkp, skp, pkn, ktunnel, x, N, halfkeyB, Exp(halfkeyB,x))

SERVER_0'(P, ksms, M, pkp, skp, pkn, ktunnel, x, N, halfkeyB, ktunnelDH) =
    true and member((Msg8, Encrypt.(ktunnelDH, <ksms, N>),<ktunnel, ksms, ktunnelDH>), OUTPUT_INT_MSG8) &
    output.P.N.(Msg8, Encrypt.(ktunnelDH, <ksms, N>),<ktunnel, ksms, ktunnelDH>) ->
    SKIP

  SERVER_1(P, ksms, M, pkp, skp, pkn, ktunnel, x) = SERVER_0(P, ksms, M, pkp, skp, pkn, ktunnel, x)

  SERVER(P, ksms, M, pkp, skp, pkn, ktunnel, x) =
    SERVER_1(P, ksms, M, pkp, skp, pkn, ktunnel, x)
      [[input.N.P.(l_,m_,re_) <- receive.N.P.ALGEBRA_M::rmb((l_,m_,re_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO3]]
      [[input.N.P.(l_,m_,re_) <- receive.N.P.ALGEBRA_M::rmb((l_,m_,re_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO5]]
      [[input.N.P.(l_,m_,re_) <- receive.N.P.ALGEBRA_M::rmb((l_,m_,re_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO7]]
      [[output.P.N.(l_,m_,se_) <- send.P.N.ALGEBRA_M::rmb((l_,m_,se_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO4]]
      [[output.P.N.(l_,m_,se_) <- send.P.N.ALGEBRA_M::rmb((l_,m_,se_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO6]]
      [[output.P.N.(l_,m_,se_) <- send.P.N.ALGEBRA_M::rmb((l_,m_,se_)) |
          N <- Agent, (l_,m_,se_,re_) <- INT_MSG_INFO8]]

  -- Messages as they appear on the network; each messages is renamed
  -- (by rmb) to the representative member of its equivalence class

  INPUT_MSG1 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG1}
  INPUT_MSG2 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG2}
  INPUT_MSG3 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG3}
  INPUT_MSG4 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG4}
  INPUT_MSG5 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG5}
  INPUT_MSG6 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG6}
  INPUT_MSG7 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG7}
  INPUT_MSG8 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG8}
  INPUT_MSG9 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG9}
  INPUT_MSG10 = {ALGEBRA_M::rmb(m_) | m_ <- INPUT_INT_MSG10}

  OUTPUT_MSG1 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG1}
  OUTPUT_MSG2 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG2}
  OUTPUT_MSG3 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG3}
  OUTPUT_MSG4 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG4}
  OUTPUT_MSG5 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG5}
  OUTPUT_MSG6 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG6}
  OUTPUT_MSG7 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG7}
  OUTPUT_MSG8 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG8}
  OUTPUT_MSG9 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG9}
  OUTPUT_MSG10 = {ALGEBRA_M::rmb(m_) | m_ <- OUTPUT_INT_MSG10}

  DIRECT_MSG1 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO1}
  DIRECT_MSG2 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO2}
  DIRECT_MSG3 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO3}
  DIRECT_MSG4 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO4}
  DIRECT_MSG5 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO5}
  DIRECT_MSG6 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO6}
  DIRECT_MSG7 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO7}
  DIRECT_MSG8 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO8}
  DIRECT_MSG9 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO9}
```

```
  DIRECT_MSG10 = {ALGEBRA_M::rmb4(m_) | m_ <- INT_MSG_INFO10}

  -- Process representing MobileNode

  Alpha_RESPONDER_MobileNode =
    Union({
      {|send.MobileNode.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG2|},
      {|send.MobileNode.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG10|},
      {|receive.A_.MobileNode.m_   |   A_   <-   ALL_PRINCIPALS,   m_   <-
INPUT_MSG1|},
      {|receive.A_.MobileNode.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG9|}
    })

  RESPONDER_MobileNode = RESPONDER(MobileNode, KSMS, PrevAR)

  Alpha_MobileNode =
    Union({
      {|env.MobileNode|},
      {|send.MobileNode.A_, receive.A_.MobileNode | A_ <- ALL_PRINCIPALS|}
    })

  AGENT_MobileNode =
    (RESPONDER_MobileNode [Alpha_RESPONDER_MobileNode || {} ] STOP)

  -- Process representing NewAR

  Alpha_INITIATOR_NewAR =
    Union({
      {|env.NewAR.m_ | m_ <- ENV_MSG0|},
      {|send.NewAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG1|},
      {|send.NewAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG3|},
      {|send.NewAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG5|},
      {|send.NewAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG7|},
      {|send.NewAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG9|},
      {|receive.A_.NewAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG2|},
      {|receive.A_.NewAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG4|},
      {|receive.A_.NewAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG6|},
      {|receive.A_.NewAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG8|},
      {|receive.A_.NewAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG10|}
    })

  INITIATOR_NewAR = INITIATOR(NewAR, PKN, SKN, Y, PKP, Na)

  Alpha_NewAR =
    Union({
      {|env.NewAR|},
      {|send.NewAR.A_, receive.A_.NewAR | A_ <- ALL_PRINCIPALS|}
    })

  AGENT_NewAR =
    (INITIATOR_NewAR [Alpha_INITIATOR_NewAR || {} ] STOP)

  -- Process representing PrevAR

  Alpha_SERVER_PrevAR =
    Union({
      {|send.PrevAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG4|},
      {|send.PrevAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG6|},
      {|send.PrevAR.A_.m_ | A_ <- ALL_PRINCIPALS, m_ <- OUTPUT_MSG8|},
      {|receive.A_.PrevAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG3|},
      {|receive.A_.PrevAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG5|},
      {|receive.A_.PrevAR.m_ | A_ <- ALL_PRINCIPALS, m_ <- INPUT_MSG7|}
    })

  SERVER_PrevAR = SERVER(PrevAR, KSMS, MobileNode, PKP, SKP, PKN, KTUNNEL,
X)

  Alpha_PrevAR =
    Union({
      {|env.PrevAR|},
      {|send.PrevAR.A_, receive.A_.PrevAR | A_ <- ALL_PRINCIPALS|}
    })

  AGENT_PrevAR =
    (SERVER_PrevAR [Alpha_SERVER_PrevAR || {} ] STOP)
```

```
exports

  ENV_MSG = {ALGEBRA_M::rmb(m_) | m_ <- ENV_INT_MSG}
  INT_MSG_INFO =
    Union({
      INT_MSG_INFO1,
      INT_MSG_INFO2,
      INT_MSG_INFO3,
      INT_MSG_INFO4,
      INT_MSG_INFO5,
      INT_MSG_INFO6,
      INT_MSG_INFO7,
      INT_MSG_INFO8,
      INT_MSG_INFO9,
      INT_MSG_INFO10
    })
  INPUT_MSG =
    Union({
      INPUT_MSG1,
      INPUT_MSG2,
      INPUT_MSG3,
      INPUT_MSG4,
      INPUT_MSG5,
      INPUT_MSG6,
      INPUT_MSG7,
      INPUT_MSG8,
      INPUT_MSG9,
      INPUT_MSG10
    })
  OUTPUT_MSG =
    Union({
      OUTPUT_MSG1,
      OUTPUT_MSG2,
      OUTPUT_MSG3,
      OUTPUT_MSG4,
      OUTPUT_MSG5,
      OUTPUT_MSG6,
      OUTPUT_MSG7,
      OUTPUT_MSG8,
      OUTPUT_MSG9,
      OUTPUT_MSG10
    })
  DIRECT_MSG =
  Union({
    DIRECT_MSG1,
    DIRECT_MSG2,
    DIRECT_MSG3,
    DIRECT_MSG4,
    DIRECT_MSG5,
    DIRECT_MSG6,
    DIRECT_MSG7,
    DIRECT_MSG8,
    DIRECT_MSG9,
    DIRECT_MSG10
  })

  channel input:ALL_PRINCIPALS.ALL_PRINCIPALS.INPUT_INT_MSG
  channel output: ALL_PRINCIPALS.ALL_PRINCIPALS.OUTPUT_INT_MSG
  channel env_I : ALL_PRINCIPALS.ENV_INT_MSG

  -- Complete system

  SYSTEM_0 =
    (AGENT_MobileNode
    |||
    (AGENT_NewAR
    |||
    AGENT_PrevAR))

endmodule

-- ***************************************************************************
-- *                               Algebra                                   *
-- ***************************************************************************

module ALGEBRA_M
```

```
  -- Algebraic laws, defined as a set of pairs

  laws =
    Union({
      {(Encrypt.(Exp(Exp(Gen, Num_1), Num_2), <ksms, N>),
        Encrypt.(Exp(Exp(Gen, Num_2), Num_1), <ksms, N>)) |
          N <- Agent, Num_1 <- Num, Num_2 <- Num, ksms <- SessionKey},
      {(Exp(Exp(Gen, Num_1), Num_2),
        Exp(Exp(Gen, Num_2), Num_1)) |
          Num_1 <- Num, Num_2 <- Num},
      {(Exp(Exp(Gen, Num_2), Num_1),
        Exp(Exp(Gen, Num_1), Num_2)) |
          Num_1 <- Num, Num_2 <- Num}
    })

  -- Calculate transitive closure of algebraic laws, and select
  -- representative member of each equivalence class

  external mtransclose
  renaming = mtransclose(laws, Fact_1)
  ren = relational_inverse_image(renaming)

  -- function that renames non-sequential fact to representative member

  applyRenaming0(a_) =
    let S_ = ren(a_)
    within if card(S_)==0 then a_ else elsing(S_)

  elsing({x_}) = x_

  domain = {a_ | (_,a_) <- renaming}

exports

  -- function that renames arbitrary fact to representative member

  applyRenaming(Sq.ms_) =
    if member(Sq.ms_, Fact_1) then applyRenaming0(Sq.ms_)
    else Sq.<applyRenaming0(m_) | m_ <- ms_>
  applyRenaming(a_) = applyRenaming0(a_)

  -- function that renames (label, fact, extras) triples

  rmb((l_,m_,extras_)) =
    (l_, applyRenaming(m_), applyRenamingToSeq(extras_))
  rmb4((l_,m_,s_extras_,r_extras_)) =
    (l_, applyRenaming(m_), applyRenamingToSeq(s_extras_),
     applyRenamingToSeq(r_extras_))

  -- lift renaming to sets and to deductions

  applyRenamingToSet(X_) =
    union({elsing(ren(a_)) | a_ <- inter(X_,domain)},  diff(X_, domain))

  applyRenamingToSeq(X_) = <applyRenaming(e_) | e_ <- X_>

  applyRenamingToDeductions(S_) =
    {(applyRenaming0(f_), applyRenamingToSet(X_)) | (f_,X_) <- S_}

endmodule

-- ***********************************************************************
-- *                             The Intruder                            *
-- ***********************************************************************

module INTRUDER_M

  Generations = DI_M::Generations

  -- Intruder's initial knowledge

  IK0_init = {MobileNode, NewAR, Mallory, PKN, PKP, PrevAR, PKMALLORY,
              SKMALLORY, Z, NInt, Gen, Garbage}

  IK0 = IK0_init
```

```
-- Intruder's deductions

unSq_ (Sq.ms_) = set(ms_)
unSq_ (m_) = {m_}

unknown_(S_) = diff(S_, IK0_init)

Base_Deductions =
  Union({SqDeductions, UnSqDeductions,
         EncryptionDeductions, DecryptionDeductions,
         VernEncDeductions, VernDecDeductions,
         FnAppDeductions, HashDeductions, UserDeductions})

SqDeductions =
  {(Sq.fs_, unknown_(set(fs_))) | Sq.fs_ <- Fact_1}

UnSqDeductions =
  {(f_, unknown_({Sq.fs_})) | Sq.fs_ <- Fact_1, f_ <- unknown_(set(fs_))}

EncryptionDeductions =
  {(Encrypt.(k_,fs_), unknown_(union({k_}, set(fs_)))) |
      Encrypt.(k_,fs_) <- Fact_1}

DecryptionDeductions =
  {(f_, unknown_({Encrypt.(k_,fs_), inverse(k_)})) |
      Encrypt.(k_,fs_) <- Fact_1, f_ <- unknown_(set(fs_))}

VernEncDeductions =
  {(Xor.(m1_,m2_), unknown_(union(unSq_(m1_), unSq_(m2_)))) |
      Xor.(m1_,m2_) <- Fact_1}

VernDecDeductions =
    {(m11_, union(unknown_(unSq_(m2_)), {Xor.(m1_,m2_)})) |
       Xor.(m1_,m2_) <- Fact_1, m11_ <- unSq_(m1_)}

HashDeductions = {(Hash.(f_, ms_), set(ms_)) | Hash.(f_, ms_) <- Fact_1}
```

```
  UserDeductions = {}

  FnAppDeductions =
    {(Exp__.(arg_1_, arg_2_), unknown_({arg_1_, arg_2_})) |
        Exp__.(arg_1_, arg_2_) <- Fact_1}

  -- close up intruder's initial knowledge under deductions;
  -- calculate which facts cannot be learnt

  components_(Sq.ms_) =
    if member(Sq.ms_, Fact_1) then {Sq.ms_} else set(ms_)
  components_(m_) = {m_}

  Seeable_ =
    Union({unknown_(components_(m_))         |         (_,m_,_,_)         <-
SYSTEM_M::INT_MSG_INFO})

  Close_(IK_, ded_, fact_) =
    let IK1_ =
          union(IK_, {f_ | (f_,fs_) <- ded_, fs_ <= IK_})
        ded1_ =
          {(f_,fs_) | (f_,fs_) <- ded_, not (member(f_,IK_)),
                      fs_ <= fact_}
        fact1_ = Union({IK_, {f_ | (f_,fs_) <- ded_}, Seeable_})
    within
    if card(IK_)==card(IK1_) and card(ded_)==card(ded1_)
       and card(fact_)==card(fact1_)
    then (IK_, {(f_,diff(fs_,IK_)) | (f_,fs_) <- ded_}, fact_)
    else Close_(IK1_, ded1_, fact1_)

  (IK1, Deductions, KnowableFact) =
    Close_(ALGEBRA_M::applyRenamingToSet(IK0),
           ALGEBRA_M::applyRenamingToDeductions(Base_Deductions),
           ALGEBRA_M::applyRenamingToSet(Fact_1))

  LearnableFact = diff(KnowableFact, IK1)

  Deductions' =  -- Don't you hate hacks like this?
```

```
    if Deductions=={} then {(Garbage,{Garbage})} else Deductions

  -- The intruder

  -- * leak is used to signal that a possible secret has been learnt
  -- * hear and say are used to represent hearing or saying a message
  -- * infer(f,fs) represent deducing fact f from the set of facts fs

  -- Types of sender and receiver of each message

  SenderType (Msg1) = Agent
  SenderType (Msg2) = Agent
  SenderType (Msg3) = Agent
  SenderType (Msg4) = Agent
  SenderType (Msg5) = Agent
  SenderType (Msg6) = Agent
  SenderType (Msg7) = Agent
  SenderType (Msg8) = Agent
  SenderType (Msg9) = Agent
  SenderType (Msg10) = Agent

  ReceiverType(Msg1) = Agent
  ReceiverType(Msg2) = Agent
  ReceiverType(Msg3) = Agent
  ReceiverType(Msg4) = Agent
  ReceiverType(Msg5) = Agent
  ReceiverType(Msg6) = Agent
  ReceiverType(Msg7) = Agent
  ReceiverType(Msg8) = Agent
  ReceiverType(Msg9) = Agent
  ReceiverType(Msg10) = Agent

  -- Component of intruder for currently unknown fact f_:
  -- * ms_ is the set of messages that contain f_ at the top level
  -- * fss_ is the set of sets of facts from which f_ can be deduced
  -- * ds_ is the set of deductions that use f_

  IGNORANT(f_,ms_,fss_,ds_) =
    hear?m_:ms_ -> KNOWS(f_,ms_,ds_)
    []
    ([] fs_ : fss_, not(member(f_,fs_)) @
        infer.(f_,fs_) -> KNOWS(f_,ms_,ds_))

  -- Component of intruder for known fact f_

  KNOWS(f_,ms_,ds_) =
    hear?m_:ms_ -> KNOWS(f_,ms_,ds_)
    []
    say?m_:ms_ -> KNOWS(f_,ms_,ds_)
    []
    ([] ded@@(f1_,fs_) : ds_, f1_!=f_ @ infer.ded -> KNOWS(f_,ms_,ds_))
    []
    member(f_,ALL_SECRETS_DI) & leak.f_ -> KNOWS(f_,ms_,ds_)

  -- Alphabet of this component

  AlphaL(f_,ms_,fss_,ds_) =
    Union({(if member(f_,ALL_SECRETS_DI) then {leak.f_} else {}),
           {hear.m_, say.m_ | m_ <- ms_},
           {infer.(f_,fs_) | fs_ <- fss_},
           {infer.(f1_,fs_) | (f1_,fs_) <- ds_}
         })

  -- Set of all (f_, ms_, fss_, ds_) for which intruder components
  -- must be built

  f_ms_fss_ds_s =
    let rid_ = relational_image(Deductions)
        msf_  =  relational_image({(f_,  m_)  |  m_  <-  MSG_BODY,  f_  <-
unSq_(m_)})
        xsf_ = relational_image({(f_, x_) | x_@@(_,fs_) <- Deductions,
                                            f_ <- fs_})
    within {(f_, msf_(f_), rid_(f_), xsf_(f_)) | f_ <- LearnableFact}

  -- Put components together in parallel ...
  INTRUDER_00 =
    (|| (f_,ms_,fss_,ds_) : f_ms_fss_ds_s @
         [AlphaL(f_,ms_,fss_,ds_)] IGNORANT(f_,ms_,fss_,ds_))

  INTRUDER_0 = INTRUDER_00 \ {|infer|}

  -- ... and rename events appropriately

  INTRUDER_1 =
    (chase(INTRUDER_0)
      [[ hear.m_ <- send.A_.B_.(l_,m_,se_) |
           (l_,m_,se_,re_) <- DIRECT_MSG,
           A_ <- diff(SenderType(l_),{Mallory}), B_ <- ReceiverType(l_) ]]
     [|{| hear |}|] STOP)
      [[ say.m_ <- receive.A_.B_.(l_,m_,re_) |
           (l_,m_,se_,re_) <- DIRECT_MSG,
           A_ <- SenderType(l_), B_ <- ReceiverType(l_) ]]

  -- Add in facts that are known initially

  SAY_KNOWN_0 =
    (inter(IK1, ALL_SECRETS_DI) != {} & dummy_leak -> SAY_KNOWN_0)
    [] dummy_send -> SAY_KNOWN_0
    [] dummy_receive -> SAY_KNOWN_0

  SAY_KNOWN =
    SAY_KNOWN_0
      [[ dummy_leak <- leak.f_ | f_ <- inter(IK1, ALL_SECRETS_DI) ]]
      [[ dummy_send <- send.A_.B_.(l_,m_,se_) |
           (l_,m_,se_,re_) <- DIRECT_MSG, components_(m_) <= IK1,
           A_ <- diff(SenderType(l_),{Mallory}), B_ <- ReceiverType(l_) ]]
      [[ dummy_receive <- receive.A_.B_.(l_,m_,re_) |
           (l_,m_,se_,re_) <- DIRECT_MSG, components_(m_) <= IK1,
           A_ <- SenderType(l_), B_ <- ReceiverType(l_) ]]

  STOP_SET = {| send.Mallory |}

exports
```

```
  -- Declare channels:
  channel hear, say : MSG_BODY
  channel infer : Deductions'
  channel dummy_leak, dummy_send, dummy_receive

  print IK1          -- intruder's initial knowledge
  print KnowableFact -- all facts that might be learnt
  print Deductions   -- all deductions over KnowableFact

  -- Complete intruder

  INTRUDER =
    (INTRUDER_1 [| STOP_SET |] STOP) ||| SAY_KNOWN

endmodule

IntruderInterface = {| send, receive |}

SYSTEM =
  SYSTEM_M::SYSTEM_0 [| IntruderInterface |] INTRUDER_M::INTRUDER

-- *********************************************************************
-- *                    Specifications and Assertions                  *
-- *********************************************************************

module SECRET_M

  -- Specification for single secret

  SECRET_SPEC_0(s_) =
    signal.Claim_Secret?A_!s_?Bs_ ->
      (if member(Mallory, Bs_) then SECRET_SPEC_0(s_)
       else SECRET_SPEC_1(s_))
    []
    leak.s_ -> SECRET_SPEC_0(s_)

  SECRET_SPEC_1(s_) =
    signal.Claim_Secret?A_!s_?Bs_ -> SECRET_SPEC_1(s_)

  -- Specification for all secrets

  AlphaS(s_) =
    Union({
      {|signal.Claim_Secret.A_.s_ | A_ <- ALL_PRINCIPALS|},
      {leak.s_}
  })

  -- Sequential version; secs_ is secrets that intruder must not learn

  SEQ_SECRET_SPEC_0(secs_) =
    scs?s_!IntIn -> SEQ_SECRET_SPEC_0(secs_)
    []
    card(secs_)<3 & scs?s_!IntNotIn ->
      SEQ_SECRET_SPEC_0(union(secs_,{s_}))
    []
    card(secs_)==3 & scs?s_:secs_!IntNotIn ->
      SEQ_SECRET_SPEC_0(secs_)
    []
    leak?s_ : diff(ALL_SECRETS,secs_) -> SEQ_SECRET_SPEC_0(secs_)

  isIntIn(S_) = if member(Mallory,S_) then IntIn else IntNotIn

  Alpha_SECRETS =
    Union({
      {|leak, signal.Claim_Secret.A_ | A_ <- HONEST|}
    })

  Alpha_SEQ_SECRETS =
    Union({
      {|leak, scs|}
    })

exports

  SECRET_SPEC = (|| s_ : ALL_SECRETS @ [AlphaS(s_)] SECRET_SPEC_0(s_))
```

```
datatype IncInt = IntIn | IntNotIn

channel scs : ALL_SECRETS.IncInt

SEQ_SECRET_SPEC = SEQ_SECRET_SPEC_0({})

-- System for secrecy checking

SYSTEM_S =
  let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
      SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
      Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
      Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
  within
  SYSTEM
    [[send.P.N.ALGEBRA_M::rmb((Msg8,   Encrypt.(ktunnelDH,   <ksms,   N>),
<ktunnel, ksms, ktunnelDH>))
        <- signal.Claim_Secret.P.ALGEBRA_M::applyRenaming(ktunnel).{N},
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- signal.Claim_Secret.N.ALGEBRA_M::applyRenaming(ktunnel).{P},
      send.M.N.ALGEBRA_M::rmb((Msg10,  Encrypt.(ksms,  <na,  M,  N>),  <na,
ksms>))
        <- signal.Claim_Secret.M.ALGEBRA_M::applyRenaming(ksms).{N},
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- signal.Claim_Secret.N.ALGEBRA_M::applyRenaming(ksms).{M},
      send.M.N.ALGEBRA_M::rmb((Msg10,  Encrypt.(ksms,  <na,  M,  N>),  <na,
ksms>))
        <- signal.Claim_Secret.M.ALGEBRA_M::applyRenaming(na).{N},
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- signal.Claim_Secret.N.ALGEBRA_M::applyRenaming(na).{M} |
          P <- Agent_renamed_, N <- Agent_renamed_,
          ktunnel <- SessionKey_renamed_, ksms <- SessionKey_renamed_,
          ktunnelDH <- Field_renamed_, M <- Agent_renamed_,
          na <- Nonce_renamed_
    ]] \ diff(Events,Alpha_SECRETS)
SYSTEM_S_SEQ =
  let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
      SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
      Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
      Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
  within
  SYSTEM
    [[send.P.N.ALGEBRA_M::rmb((Msg8,  Encrypt.(ktunnelDH,  <ksms,  N>),
<ktunnel, ksms, ktunnelDH>))
        <- scs.ALGEBRA_M::applyRenaming(ktunnel).isIntIn({P, N}),
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- scs.ALGEBRA_M::applyRenaming(ktunnel).isIntIn({N, P}),
      send.M.N.ALGEBRA_M::rmb((Msg10,  Encrypt.(ksms,  <na,  M,  N>),  <na,
ksms>))
        <- scs.ALGEBRA_M::applyRenaming(ksms).isIntIn({M, N}),
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- scs.ALGEBRA_M::applyRenaming(ksms).isIntIn({N, M}),
      send.M.N.ALGEBRA_M::rmb((Msg10,  Encrypt.(ksms,  <na,  M,  N>),  <na,
ksms>))
        <- scs.ALGEBRA_M::applyRenaming(na).isIntIn({M, N}),
      receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>))
        <- scs.ALGEBRA_M::applyRenaming(na).isIntIn({N, M}) |
          P <- Agent_renamed_, N <- Agent_renamed_,
          ktunnel <- SessionKey_renamed_, ksms <- SessionKey_renamed_,
          ktunnelDH <- Field_renamed_, M <- Agent_renamed_,
          na <- Nonce_renamed_
    ]] \ diff(Events,Alpha_SEQ_SECRETS)

endmodule

-- Assertion of secrecy

assert SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
assert SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ

-- Authentication specifications
-- Authentication specification number 1

module AUTH1_M

-- Spec parameterized by name of agent being authenticated

AuthenticateRESPONDERToINITIATORAgreement_na_0(M) =
  signal.Running1.RESPONDER_role.M?N?na ->
  signal.Commit1.INITIATOR_role.N.M.na -> STOP

AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(M) =
  {|signal.Running1.RESPONDER_role.M.N,
    signal.Commit1.INITIATOR_role.N.M |
       N <- inter(Agent, HONEST)|}

-- Specs for particular agents being authenticated

AuthenticateRESPONDERMobileNodeToINITIATORAgreement_na =
  AuthenticateRESPONDERToINITIATORAgreement_na_0(MobileNode)

AuthenticateRESPONDERNewARToINITIATORAgreement_na =
  STOP

AuthenticateRESPONDERPrevARToINITIATORAgreement_na =
  STOP

-- alphabet of specification

alphaAuthenticateRESPONDERToINITIATORAgreement_na =
  Union({
    AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(MobileNode),
    AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(NewAR),
    AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(PrevAR)
  })

exports

-- Specs for all agents being authenticated
AuthenticateRESPONDERToINITIATORAgreement_na =
   (AuthenticateRESPONDERMobileNodeToINITIATORAgreement_na
   [|
inter(AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(MobileNode),

union(AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(NewAR),
AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(PrevAR))) |]
   (AuthenticateRESPONDERNewARToINITIATORAgreement_na
   [| inter(AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(NewAR),
            AlphaAuthenticateRESPONDERToINITIATORAgreement_na_0(PrevAR))
|]
   AuthenticateRESPONDERPrevARToINITIATORAgreement_na))

-- System for authentication checking

SYSTEM_1 =
   let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
       Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
       SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
       Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
   within
   SYSTEM
     [[send.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
ksms>)) <-
          signal.Running1.RESPONDER_role.M.N.na,
        receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>)) <-
          signal.Commit1.INITIATOR_role.N.M.na |
            M <- Agent_renamed_, N <- Agent_renamed_, na <- Nonce_renamed_,
            ksms <- SessionKey_renamed_, P <- Agent_renamed_,
            ktunnel <- SessionKey_renamed_, ktunnelDH <- Field_renamed_
     ]]
     \ diff(Events, alphaAuthenticateRESPONDERToINITIATORAgreement_na)

endmodule

assert AUTH1_M::AuthenticateRESPONDERToINITIATORAgreement_na [T=
     AUTH1_M::SYSTEM_1
```

```
-- Authentication specification number 2

module AUTH2_M

  -- Spec parameterized by name of agent being authenticated

  AuthenticateINITIATORToRESPONDERAgreement_ksms_0(N) =
    signal.Running2.INITIATOR_role.N?M?ksms ->
    signal.Commit2.RESPONDER_role.M.N.ksms -> STOP

  AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(N) =
    {|signal.Running2.INITIATOR_role.N.M,
      signal.Commit2.RESPONDER_role.M.N |
         M <- inter(Agent, HONEST)|}

  -- Specs for particular agents being authenticated

  AuthenticateINITIATORMobileNodeToRESPONDERAgreement_ksms =
    STOP

  AuthenticateINITIATORNewARToRESPONDERAgreement_ksms =
    AuthenticateINITIATORToRESPONDERAgreement_ksms_0(NewAR)

  AuthenticateINITIATORPrevARToRESPONDERAgreement_ksms =
    STOP

  -- alphabet of specification

  alphaAuthenticateINITIATORToRESPONDERAgreement_ksms =
    Union({
      AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(MobileNode),
      AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(NewAR),
      AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(PrevAR)
    })

exports

  -- Specs for all agents being authenticated
  AuthenticateINITIATORToRESPONDERAgreement_ksms =
    (AuthenticateINITIATORMobileNodeToRESPONDERAgreement_ksms
    [|
inter(AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(MobileNode),

union(AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(NewAR),
AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(PrevAR))) |]
    (AuthenticateINITIATORNewARToRESPONDERAgreement_ksms
    [| inter(AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(NewAR),
             AlphaAuthenticateINITIATORToRESPONDERAgreement_ksms_0(PrevAR))
|]
    AuthenticateINITIATORPrevARToRESPONDERAgreement_ksms))

  -- System for authentication checking

  SYSTEM_2 =
    let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
        SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
        Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
    within
    SYSTEM
      [[send.N.M.ALGEBRA_M::rmb((Msg9, Encrypt.(ksms, <na, N, M>), <ksms>))
<-
          signal.Running2.INITIATOR_role.N.M.ksms,
        send.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
ksms>)) <-
          signal.Commit2.RESPONDER_role.M.N.ksms |
            N <- Agent_renamed_, M <- Agent_renamed_,
            ksms <- SessionKey_renamed_, na <- Nonce_renamed_
      ]]
      \ diff(Events, alphaAuthenticateINITIATORToRESPONDERAgreement_ksms)

endmodule

assert AUTH2_M::AuthenticateINITIATORToRESPONDERAgreement_ksms [T=
       AUTH2_M::SYSTEM_2

-- Authentication specification number 3
```

```
module AUTH3_M

  -- Spec parameterized by name of agent being authenticated

  AuthenticateSERVERToINITIATORAgreement_ktunnel_0(P) =
    signal.Running3.SERVER_role.P?N?ktunnel ->
    signal.Commit3.INITIATOR_role.N.P.ktunnel -> STOP

  AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(P) =
    {|signal.Running3.SERVER_role.P.N,
      signal.Commit3.INITIATOR_role.N.P |
         N <- inter(Agent, HONEST)|}

  -- Specs for particular agents being authenticated

  AuthenticateSERVERMobileNodeToINITIATORAgreement_ktunnel =
    STOP

  AuthenticateSERVERNewARToINITIATORAgreement_ktunnel =
    STOP

  AuthenticateSERVERPrevARToINITIATORAgreement_ktunnel =
    AuthenticateSERVERToINITIATORAgreement_ktunnel_0(PrevAR)

  -- alphabet of specification

  alphaAuthenticateSERVERToINITIATORAgreement_ktunnel =
    Union({
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(MobileNode),
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(NewAR),
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(PrevAR)
    })

exports

  -- Specs for all agents being authenticated

  AuthenticateSERVERToINITIATORAgreement_ktunnel =
    (AuthenticateSERVERMobileNodeToINITIATORAgreement_ktunnel
    [|
inter(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(MobileNode),

union(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(NewAR),
AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(PrevAR))) |]
    (AuthenticateSERVERNewARToINITIATORAgreement_ktunnel
    [| inter(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(NewAR),
            AlphaAuthenticateSERVERToINITIATORAgreement_ktunnel_0(PrevAR))
|]
    AuthenticateSERVERPrevARToINITIATORAgreement_ktunnel))


  -- System for authentication checking


  SYSTEM_3 =
    let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
        SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
        Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
        Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
    within
    SYSTEM
      [[send.P.N.ALGEBRA_M::rmb((Msg8,   Encrypt.(ktunnelDH,   <ksms,   N>),
<ktunnel, ksms, ktunnelDH>)) <-
          signal.Running3.SERVER_role.P.N.ktunnel,
        receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>)) <-
          signal.Commit3.INITIATOR_role.N.P.ktunnel |
            P <- Agent_renamed_, N <- Agent_renamed_,
            ktunnel <- SessionKey_renamed_, ksms <- SessionKey_renamed_,
            ktunnelDH <- Field_renamed_, na <- Nonce_renamed_,
            M <- Agent_renamed_
      ]]
      \ diff(Events, alphaAuthenticateSERVERToINITIATORAgreement_ktunnel)


endmodule


assert AUTH3_M::AuthenticateSERVERToINITIATORAgreement_ktunnel [T=
       AUTH3_M::SYSTEM_3


-- Authentication specification number 4
```

```
module AUTH4_M

  -- Spec parameterized by name of agent being authenticated

  AuthenticateSERVERToINITIATORAgreement_ksms_0(P) =
    signal.Running4.SERVER_role.P?N?ksms ->
    signal.Commit4.INITIATOR_role.N.P.ksms -> STOP

  AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(P) =
    {|signal.Running4.SERVER_role.P.N,
      signal.Commit4.INITIATOR_role.N.P |
         N <- inter(Agent, HONEST)|}

  -- Specs for particular agents being authenticated

  AuthenticateSERVERMobileNodeToINITIATORAgreement_ksms =
    STOP

  AuthenticateSERVERNewARToINITIATORAgreement_ksms =
    STOP

  AuthenticateSERVERPrevARToINITIATORAgreement_ksms =
    AuthenticateSERVERToINITIATORAgreement_ksms_0(PrevAR)

  -- alphabet of specification

  alphaAuthenticateSERVERToINITIATORAgreement_ksms =
    Union({
      AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(MobileNode),
      AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(NewAR),
      AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(PrevAR)
    })

exports

  -- Specs for all agents being authenticated
  AuthenticateSERVERToINITIATORAgreement_ksms =
    (AuthenticateSERVERMobileNodeToINITIATORAgreement_ksms
    [|
inter(AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(MobileNode),

union(AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(NewAR),
AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(PrevAR))) |]
    (AuthenticateSERVERNewARToINITIATORAgreement_ksms
    [| inter(AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(NewAR),
            AlphaAuthenticateSERVERToINITIATORAgreement_ksms_0(PrevAR)) |]
    AuthenticateSERVERPrevARToINITIATORAgreement_ksms))


  -- System for authentication checking


  SYSTEM_4 =
    let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
        SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
        Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
        Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
    within
    SYSTEM
      [[send.P.N.ALGEBRA_M::rmb((Msg8,   Encrypt.(ktunnelDH,   <ksms,   N>),
<ktunnel, ksms, ktunnelDH>)) <-
          signal.Running4.SERVER_role.P.N.ksms,
        receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>)) <-
          signal.Commit4.INITIATOR_role.N.P.ksms |
            P <- Agent_renamed_, N <- Agent_renamed_,
            ktunnel <- SessionKey_renamed_, ksms <- SessionKey_renamed_,
            ktunnelDH <- Field_renamed_, na <- Nonce_renamed_,
            M <- Agent_renamed_
      ]]
      \ diff(Events, alphaAuthenticateSERVERToINITIATORAgreement_ksms)


endmodule


assert AUTH4_M::AuthenticateSERVERToINITIATORAgreement_ksms [T=
       AUTH4_M::SYSTEM_4


-- Authentication specification number 5
```

```
module AUTH5_M

  -- Spec parameterized by name of agent being authenticated

  AuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(P) =
    signal.Running5.SERVER_role.P?N?ktunnelDH ->
    signal.Commit5.INITIATOR_role.N.P.ktunnelDH -> STOP

  AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(P) =
    {|signal.Running5.SERVER_role.P.N,
      signal.Commit5.INITIATOR_role.N.P |
         N <- inter(Agent, HONEST)|}

  -- Specs for particular agents being authenticated

  AuthenticateSERVERMobileNodeToINITIATORAgreement_ktunnelDH =
    STOP

  AuthenticateSERVERNewARToINITIATORAgreement_ktunnelDH =
    STOP

  AuthenticateSERVERPrevARToINITIATORAgreement_ktunnelDH =
    AuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(PrevAR)

  -- alphabet of specification

  alphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH =
    Union({
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(MobileNode),
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(NewAR),
      AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(PrevAR)
    })

exports

  -- Specs for all agents being authenticated
  AuthenticateSERVERToINITIATORAgreement_ktunnelDH =
    (AuthenticateSERVERMobileNodeToINITIATORAgreement_ktunnelDH
    [|
inter(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(MobileNode),

union(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(NewAR),
AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(PrevAR))) |]
    (AuthenticateSERVERNewARToINITIATORAgreement_ktunnelDH
    [|
inter(AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(NewAR),

AlphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH_0(PrevAR)) |]
    AuthenticateSERVERPrevARToINITIATORAgreement_ktunnelDH))


  -- System for authentication checking


  SYSTEM_5 =
    let Agent_renamed_ = ALGEBRA_M::applyRenamingToSet(Agent)
        SessionKey_renamed_ = ALGEBRA_M::applyRenamingToSet(SessionKey)
        Field_renamed_ = ALGEBRA_M::applyRenamingToSet(Field)
        Nonce_renamed_ = ALGEBRA_M::applyRenamingToSet(Nonce)
    within
    SYSTEM
      [[send.P.N.ALGEBRA_M::rmb((Msg8,   Encrypt.(ktunnelDH,   <ksms,   N>),
<ktunnel, ksms, ktunnelDH>)) <-
          signal.Running5.SERVER_role.P.N.ktunnelDH,
        receive.M.N.ALGEBRA_M::rmb((Msg10, Encrypt.(ksms, <na, M, N>), <na,
P, ktunnel, ksms, ktunnelDH>)) <-
          signal.Commit5.INITIATOR_role.N.P.ktunnelDH |
            P <- Agent_renamed_, N <- Agent_renamed_,
            ktunnel <- SessionKey_renamed_, ksms <- SessionKey_renamed_,
            ktunnelDH <- Field_renamed_, na <- Nonce_renamed_,
            M <- Agent_renamed_
      ]]
      \ diff(Events, alphaAuthenticateSERVERToINITIATORAgreement_ktunnelDH)


endmodule


assert AUTH5_M::AuthenticateSERVERToINITIATORAgreement_ktunnelDH [T=
       AUTH5_M::SYSTEM_5
```

Curriculum Vitae

Rene Alexander Soltwisch

Geboren	1.3.1974 in Lübeck
Staatsangehörigkeit	deutsch
Familienstand	ledig

Wissenschaftlicher Werdegang

1990 – 1995	Oberschule zum Dom
WS 1996 – WS 2002/2003	Studium der Informatik an der Medizinischen Universität zu Lübeck
1.4.1999 – 30.6.2000	Studentische Hilfskraft in Institut für Telematik
28.7.2000 – 30.3.2001	KDD R&D Laboratories Inc. Tokyo/Japan Network Management System Laboratories
seit 15.03.2003	Wissenschaftlicher Mitarbeiter & Doktorand an der Georg-August-Universität zu Göttingen. Institut für Informatik