ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Diploma Thesis
DA-2003.01

Non-Repudiation of Service Consumption in MobyDick Networks

Winter Term 2002/2003
March 13, 2003

Egon Burgener

Tutor: Hasan
Supervisor: Prof. Dr. Burkhard Stiller

Zusammenfassung

Das vorliegende Dokument präsentiert eine Lösung zum Problem der Beweisführung des Dienstkonsums. Der Netzwerkbetreiber soll die Möglichkeit haben, die Richtigkeit der Verrechnung beweisen zu können. Falls er den Dienstkonsum nicht beweisen kann, muss der Kunde die Rechnung nicht bezahlen. Daher ist es nicht nötig, dass der Kunde irgendwelche Beweisstücke sichern muss. Ausserdem sind manche Geräte der Kunden dazu nicht fähig. Die Lösung ist speziell auf die MobyDick Architektur zugeschnitten, kann aber durch wenige Veränderungen auf andere Architekturen übernommen werden. Die MobyDick Architektur wird um drei Entitäten dem Non-Repudiation Server, den Non-Repudiation Relay Agents und den Non-Repudiation Clients erweitert.
Die Beweisstücke werden vom Gerät des Kunden generiert und mittels des Non-Repudiation Protokolls zum Non-Repudiation Server des Netzwerks transferiert. Die Beweisstücke werden vom Server verwaltet und in einer Datenbank gesichert. Es gibt zwei Typen von Beweisstücken, die Registration-Evidence-Token welche den Beweis zur Registration eines Kunden liefern und die Service-Evidence-Token welche den Beweis zur Dienstanforderung und zum Dienstkonsum liefern.
Zur Erstellung der Beweise werden asymmetrische Signaturen verwendet. Die Generierung des privaten und öffentlichen Schlüssels ist die Aufgabe des Kunden. Das vorliegende Dokument schlägt eine Zertifizierung durch eine "Trusted Third Party" (TTP) vor, präsentiert aber auch eine eingeschränkte Lösung ohne TTP.
Alternativen, mögliche Attacken und Fehlverhalten werden diskutiert.

Danksagung Ich danke meinem Betreuer Hasan für die Unterstützung und die wertvollen Ratschläge. Er stand mir stets hilfreich zur Seite.

Abstract

This document presents a solution to the problem of proving service consumption. The Service Provider should be able to give proof of the correctness of the bill. If the Service Provider isn't able to prove the service consumption the client does not have to pay for it. Therefore the client does not have to save any evidence tokens. Furthermore the clients devices may not be able to save them. The solution is especially designed for the Moby-Dick architecture and services, but can with some minor changes be adopted to any other architecture. The MobyDick architecture has to be extended with three entities, the Non-Repudiation Server, the Non-Repudiation Relay Agents and the Non-Repudiation Clients.
The evidence tokens are generated by the client's device and are transferred to the Non-Repudiation Server using the Non-Repudiation Protocol. The server manages and stores the evidence tokens in a database. There are two types of tokens, the Registration-Evidence-Tokens giving proof of registration and the Service-Evidence-Tokens giving proof of service request and service usage.
Asymmetric signatures are used to generate the evidence. It is the clients task to generate the private and public keys. The document proposes the use of a Trusted Third Party (TTP) to certify the public key, but presents also a restricted solution without a TTP.
Alternatives, possible attacks and misbehaviour are discussed.

Acknowledgment I would like to give a special thank to my tutor Hasan for the great support. He always took time to help me if needed and gave useful advice.

Contents

1 Introduction **7**
- 1.1 The MobyDick Network Architecture . . . 7
- 1.2 Non Repudiation . . . 8
- 1.3 Task Description and Goals of the Thesis . . . 9

2 Non-Repudiation in MobyDick Networks **10**
- 2.1 Requirements . . . 10
- 2.2 The Non-Repudiation Protocol . . . 10
  - 2.2.1 The Evidence Tokens . . . 11
  - 2.2.2 The Protocol . . . 11
- 2.3 The extended MobyDick Architecture . . . 15
- 2.4 The Key Management . . . 16
- 2.5 Alternatives and Optimizations . . . 16
- 2.6 Attacks or Misbehaviour . . . 17

3 Implementation **20**
- 3.1 CAAP and dia_dummy . . . 20
- 3.2 Non-Repudiation Server Software non_rep_s . . . 23
  - 3.2.1 The Database . . . 25
- 3.3 Non-Repudiation Relay Agent Software non_rep_a . . . 26
- 3.4 Non-Repudiation Client Software non_rep_c . . . 27

4 Future Work **30**

A Installation and Configuration **31**
- A.1 The CAAP software . . . 31
- A.2 The Diameter Server . . . 31
- A.3 The Non-Repudiation Software . . . 32
- A.4 Example configuration . . . 34

B Structs and Protocol Formats **36**
- B.1 Structs used by Messages . . . 36
- B.2 Structs used by dynamic Lists . . . 37
- B.3 Protocol Formats . . . 38

List of Figures

1 MobyDick Network Architecture 8
2 Time diagram of the Non-Repudiation Protocol 13
3 Extended MobyDick Architecture 15
4 Structure of pep6_s [Sch02] 20
5 Structure of dia_dummy 22
6 Structure of non_rep_s 24
7 ER diagram of database 26
8 Structure of non_rep_a 27
9 Structure of non_rep_c 28
10 MobyDick example Network 34
11 Header Format of the Main Packet Type 39
12 TLV Format 39
13 ACK Packet Format 39
14 NAK Packet Format 40

List of Tables

1 AVP List of NAS_REQ command 21
2 Diameter commands and their AVPs 23
3 Defines of dummy_s.h 32
4 Defines of non_rep_c.h 33
5 List of defined TLVs 39
6 List of defined NAK codes 40

1 Introduction

This section gives an overview on the topics this diploma thesis is based on.

1.1 The MobyDick Network Architecture

MobyDick is the name for the European Union (EU) project "Mobility and Differentiated Services in a Future IP Network". The project is carried out within the 5th framework of the Information Society Technology (IST) Program. The official project identification is IST-2000-25304 [IST02a].
The goal is to define, implement and evaluate an IPv6-based mobility-enabled end-to-end QoS architecture starting from the current IETF's QoS models, Mobile-IPv6 and AAA framework [MDP].The project started on January 1, 2001 and will last until December 31, 2003. The expected results are that the architecture supports mobile IP end-to-end communication with QoS, seamless hand-over and all necessary AAA and charging mechanisms to satisfy the user and the network operator.

The Moby Dick Network Architecture is described in [IST02b, IST02c]. The main differences to today's Internet Servive Provider networks is that clients may use mobile devices with heterogenous access technologies, that they get Quality of Service (QoS) enabled access, that seamless handover between the Access Routers is provided, and that the clients may access networks of foreign administrative domains.
The Access Routers (AR) are the points where the clients connect to the network using Ethernet, WLAN (IEEE 802.11) or the 3rd generation of mobile communication (UMTS). Since the clients may move, they may also change their Access Router. The process of changing the Access Router is called handover.
The users are identified by the Network Access Identifier (NAI). If a client wants to access the network he has to register first. The registration is needed to authenticate the client and to authorize his access. This is done by the Authentication, Authorization, Accounting and Charging Server (AAAC). The Mobile Terminal (MT) and the AR use a User Registration Protocol (URP) for the registration process. The communication between AR and AAAC Server is done by using a Authentication, Authorization and Accounting (AAA) protocol. After registration the link between MT and AR will be secured with the IPSec Authentication Header (AH).
As soon as the registration is successfully done, the client is able to use the different services according to the contract (user profile). These services are distinguished by the Differentiated Services Code Point (DSCP) field [NBBB98] in the IPv6 header which characterizes the QoS. There's a QoS Manager on each AR which controls the use of the services. If a client begins to use a new service the QoS Manager asks the QoS Broker which

decides whether a client is allowed to use the requested service. The Marker software on the MT and AR sets or recognizes the DSCP field. The DSCP flag is used to perform a service request indirectly. The client is also able to use different services at the same time.

Figure 1 presents a possible MobyDick Network.

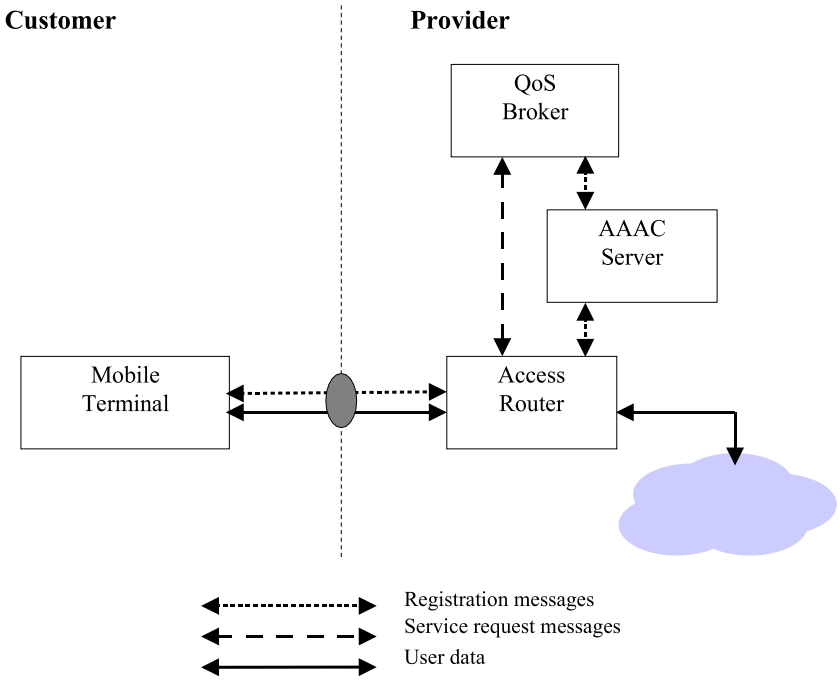

Figure 1: MobyDick Network Architecture

1.2 Non Repudiation

Non-Repudiation is a Security Service. With the help of generated evidence tokens it is possible to prove a certain action. There are different types of Non-Repudiation Services:

- Non-Repudiation of Origin (NRO).
- Non-Repudiation of Receipt (NRR).
- Non-Repudiation of Delivery (NRD).
- Non-Repudiation of Submission (NRS).

The Non-Repudiation of Origin guards against the originator of a message falsely denying having sent the message. It provides the recipient with the proof of origin. The Non-Repudiation of Receipt guards against the recipient of a message falsely denying having received the message. The Non-Repudiation of Delivery and the Non-Repudiation of Submission are supplied by the delivery agent. They provide the originator of the message with evidence that the message has been delivered and submitted to the recipient respectively [JD97].
There are two approaches in creating and collecting evidence tokens: With or without a Trusted Third Party (TTP). In early efforts, a TTP acted as a delivery agent to provide Non-Repudiation of Submission (NRS) and Non-Repudiation of Delivery (NRD). Current Non-Repudiation protocols reduced the involvement of TTPs to deal with keys only rather than with the content of transferred messages [JD96]. By using asymmetric cryptography the proof of the binding between user and public key is absolute necessary.
A Non-Repudiation protocol is fair, if it provides the originator and the recipient with valid irrefutable evidence after completion of the protocol, without giving a party an advantage over the other party in any possible incomplete protocol run [JD96]. Therefore a fair non-repudiation protocol without using a TTP isn't possible.
This diploma thesis deals with Non-Repudiation of Origin and the protocol needs not to be fair.

1.3 Task Description and Goals of the Thesis

The main task within this diploma thesis is to design and implement a Non-Repudiation protocol for registration, service request, and service consumption, that is applicable to the MobyDick architecture and services. Involvement of a TTP, if not avoidable, should be reduced to a minimum. Proving service request and access granting is a more general case, while proving service provisioning and service consumption is usually application-dependent. The metering parameters describing service consumption within MobyDick are not restricted to service duration only, but consist also of volume.

2 Non-Repudiation in MobyDick Networks

This section gives an overview of the design without details about the Implementation. First the requirements are listed. After presentation of the solution, possible alternatives and attacks are discussed.

2.1 Requirements

As mentioned the clients may use different types of devices to connect to the MobyDick Network, such as notebooks, desktop computers, mobile phones, Pads and all other mobile devices. Some of them are not capable to store a big amount of information for a long time. That this information will be stored secured and backuped isn't always possible. Therefore the client can't store any evidence tokens.
With respect to possible low computational power the amount of cryptographic computations has to be minimized.
In MobyDick different services are defined. The Non-Repudiation Protocol should consider the service with the lowest bandwith. It should not use too much bandwidth.
On the provider's side a system with well defined Interfaces is needed. To make integration easier it's important not to change the existing system too much. There should be only extensions, but no fundamental changes.
As already explained in the previous section the client has to register to the network before using any service. He is also able to use different services at the same time. As soon as the client begins to use a service for the first time the DSCP field will be set accordingly. This is known as an implicit service request. The service consumption can be divided into three parts: the Registration, the Service Request and the Service Usage. All of these parts have to be secured.

2.2 The Non-Repudiation Protocol

To provide a Non-Repudiation service it's necessary to move evidence tokens from one party to the other. Since it's not reasonable that the client stores any evidence tokens the provider has to do it. As a consequence the client is innocent as long as the provider isn't able to prove the opposite. In this way the client generates the evidence tokens and the provider has to store them. To get the evidence tokens from the clients device to the providers database a specific protocol is needed. As shown in section 1.1 "The Moby Dick Network Architecture" the clients MT communicates with the Network only through the current AR.

2.2.1 The Evidence Tokens

The service consumption consists of three parts: the registration, the service request and the service usage. The Registration-ID identifies a registration uniquely. Therefore the Registration-ID must contain the NAI and the current Session-ID. The User must have the Session-ID in order to be able to generate the evidence tokens. The evidence token of registration consists of the Registration-ID and the signature over it. It will be called the Registration-Evidence-Token.
As described in section 1.1 "The Moby Dick Network Architecture" the clients perform the service request indirectly by setting the DSCP flag in the IPv6 header. Since the provider has to be able to prove the service usage, it's necessary to generate evidence tokens periodically while using the service. The period can be based either on time or on the amount of transferred data. If the provider chooses a too long period it would be unfair, since the client signs the request of a period before its usage. On the other side if a too short period is chosen it would waste computational power and bandwidth too much.
To solve this problem and to allow shorter periods a special technique named chained hashes will be used. A hash chain consists of m hashes. The first hash is the digest of a random input r. The others will be computed as follows:

$$\begin{aligned} H_0 &= Hash(r) \\ H_n &= Hash(H_{n-1}) \\ \\ Hash &= Hashfunction \\ n &= 1...m \end{aligned}$$

It's computational impossible to calculate H_{n-1} from H_n if the random input r is unknown. To use this technique r must be unpredictable. In section 2.6 "Attacks or Misbehaviour" possible attacks on this technique will be discussed.
With the help of chained hashes it is possible to generate evidence tokens without signatures and therefore of less size. In this way there is one evidence token to provide the evidence of service request and service usage of one service type. It will be called Service-Evidence-Token. The token consists of the DSCP, the Registration-ID, the Hash H_m, the latest Hash H_n and the signature over DSCP, Registration-ID and Hash H_m.

2.2.2 The Protocol

As mentioned above the provider has to store the evidence tokens. A new entity called the Non-Repudiation Server is introduced to the architecture to collect and to store them. The Non-Repudiation Protocol is responsible to move the evidence tokens from the clients device through the Access

Router to the Non-Repudiation Server. Since the client may change the Access Router while using the services, it's the better choice to build the protocol on the UDP than on the TCP. On the other hand the TCP is reliable which makes the protocol design much easier. For this reasons the comunication between client and access router is build on UDP and the comunication between Access Router and Non-Repudiation Server is build on TCP. Keeping the connection alive saves the overhead of TCP's connection establishment. Figure 2 shows a time diagram of the exchanged Non-Repudiation packets.

If the client sends a Non-Repudiation packet to the network the Access Router sends an acknowledge (ACK) back. Since ACK may be lost and changing of the current Access Router is possible, the Access Router may not recognize duplicates. Therefore the Non-Repudiation Server has to recognize them.

The Non-Repudiation Server has to inform the network if a user has sent the evidence token to use a service or if a user doesn't send evidence tokens anymore. An interface between Non-Repudiation Server and AAAC Server is defined to exchange this information. The protocol used is Diameter [P.C02]. Diameter is a AAA protocol, but can also be used for other applications. Extendability, easy handling and reusing of AAAC Server's Diameter implementation are the main reasons for using Diameter as communication protocol to the network.

The whole communication process can be divided into three parts: the Registration, the Service Request and the Service Usage. After Registration one or more Service Requests follows. The refresh time of Service Requests should be about some minutes. Many Service Usage messages follows each Service Request. During the Registration process the Registration-Evidence-Token will be transferred to the Non-Repudiation Server. The Service Request messages carries a Service-Evidence-Token. The Service Usage process moves the latest Hash H_n to the Server which updates the corresponding Service-Evidence-Token. The following paragraphs describe the three parts in detail.

The Registration: As described in section 1.1 "The MobyDick Network Architecture" a URP is used to perform the Registration. After the client received the URP response containing the current Session-ID, the client has to send the Reg Info message.
The Reg Info packet consists of the Session-ID and NAI as the Registration-ID and the signature over it. Information about used hash and signature algorithm can be added optionally. At the same time the AAAC server sends

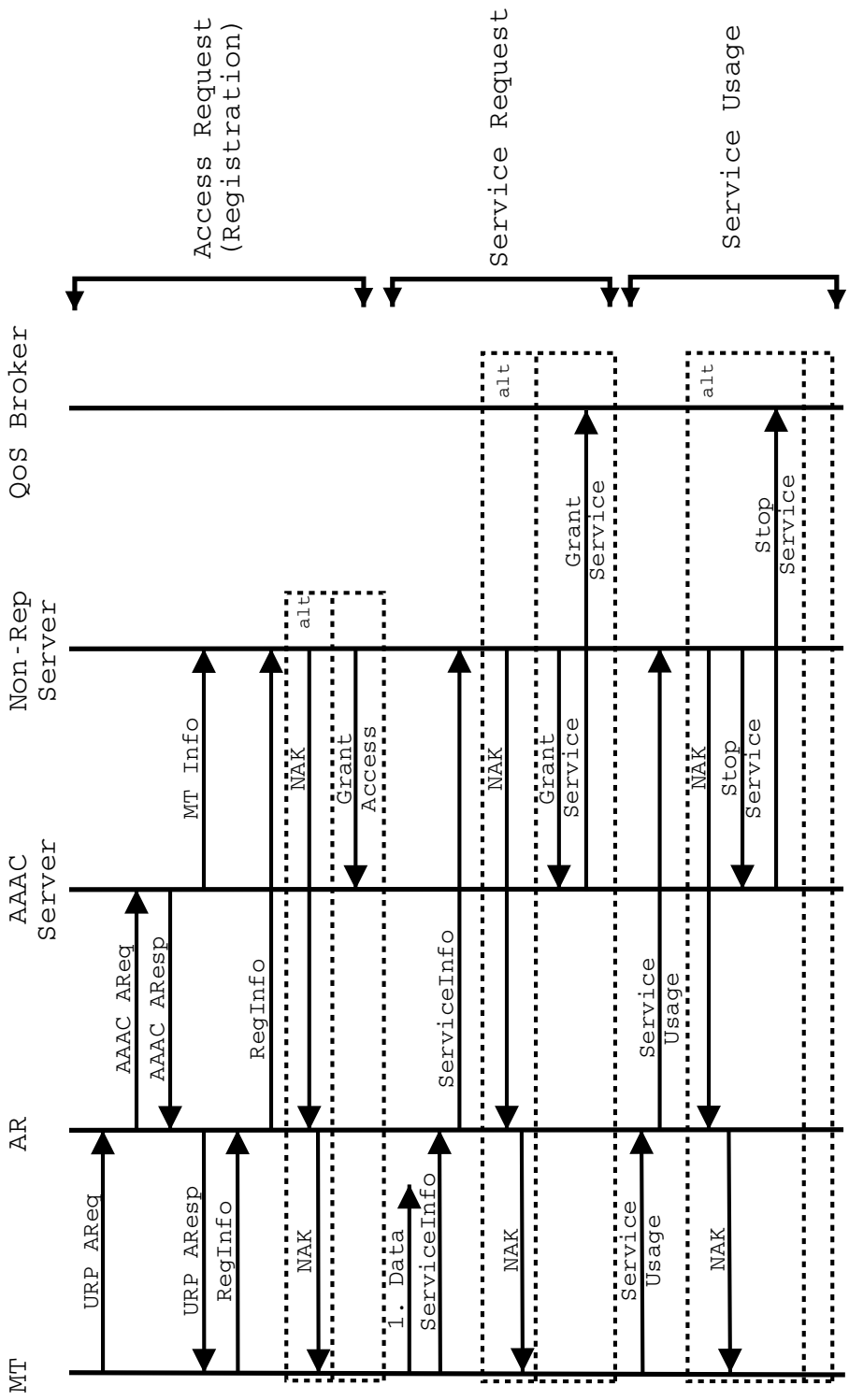

Figure 2: Time diagram of the Non-Repudiation Protocol

a MT Info message to the Non-Repudiation Server. The Message provides the Non-Repudiation Server with the information about the Registration-ID and the address of the current Access Router the client is connected to. This information about the current Access Router is necessary to know where to send the Not-Acknowledge (NAK) messages.
As soon as the Non-Repudiation Server received a Reg Info packet, it will check if the Registration-ID matches with the Registration-ID contained in the MT Info packet and verifies the signature. If one of these checks fails a NAK message will be sent to the client otherwise a Grant Access will be sent to the AAAC Server.

The Service Request: After successful registration the client is allowed to use different services. He does not have to do an explicit Service Request. Instead he begins to use the service and sets the DSCP flag in the IPv6 header. At the same time the client has to send a Service Info packet for the first time. Thenceforward the client has to send new Service Info packets periodically depending on the chosen Service Usage Period and the number of hashes m.
The Service Info packet contains the Registration-ID, the DSCP, the main hash H_m of the hashchain and the signature over all of them. If the Registration-ID matches and the signature verification was successful the server sends a Grant Service message to the AAAC Server otherwise it sends a NAK packet to the client containing the errorcode.
As soon as the AAAC Server received both the Grant Access and the Grant Service messages it will send a Grant Service message to the QoS Broker which will grant the service usage for this NAI.

The Service Usage: Within two Service Requests m Service Usage packets has to be sent periodically. The Service Usage packets are used to send the hashes of the hashchain in descending order $(H_m, H_{m-1}, \dots)$. The length of the interval is a tradeoff between fairness and bandwidth usage. The Service Usage packet consists of the Registration-ID, the DSCP and the Hash H_n. There's no need of a signature because only the real client is able to send the hash H_n. Read section 2.6 "Attacks and Misbehaviour" for more information.
If the Registration-ID mismatches or the hash verification fails, the server sends a NAK with the corresponding errorcode back to the client and a Stop Service message to the AAAC Server. The AAAC Server is responsible to inform the QoS Broker to block the service consumption of the client.

The Non-Repudiation Server stops the service consumption after the expiration of granted period if no Service Usage or no Service Info packet arrived anymore. Waiting for a short time may enhance the usability since

packets may be delayed.

The format of the packets described above are presented in the appendix B.3.

2.3 The extended MobyDick Architecture

In section 1.1 the MobyDick Architecture was presented. To add the Non-Repudiation security service it's necessary to introduce new entities: The Non-Repudiation Client on the clients device, the Non-Repudiation Relay Agent on each Access Router and the Non-Repudiation Server. Figure 3 presents the extended Architecture.

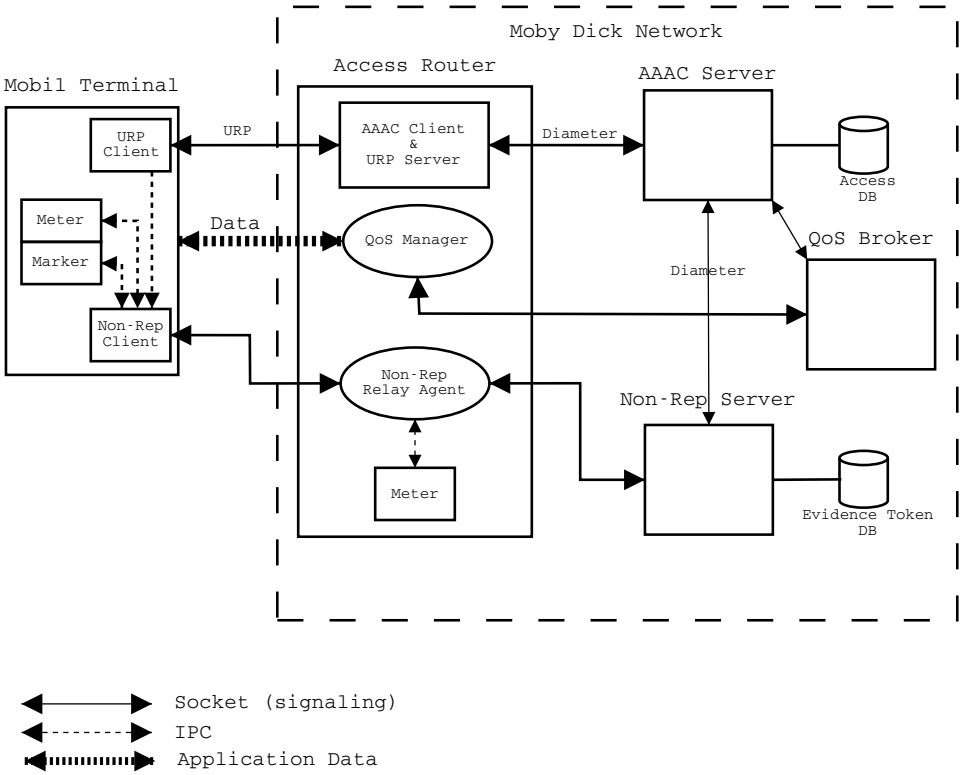

Figure 3: Extended MobyDick Architecture

Since the clients are only able to connect to the Access Router directly, the Relay Agent's task is to forward the Non-Repudiation packets to the server. The second task is to collect metering information and to send them also to the server. The metering information are needed if the period is based on the amount of transfered data. This information are collected by the meter software on each Access Router.

The server is part of the MobyDick Network and has access to the AAAC Server and to each Access Router. The received evidence tokens will be stored in a database. The interface between AAAC Server and Non-Repudiation Server is based on the Diameter protocol. The new defined commands are: MT Info, Grant Access, Grant Service and Stop Service. Each command carries one or more Attribute Value Pairs (AVP). The exact list of AVP are presented in section 3.1 “CAAP and dia_dummy”.
The Non-Repudiation Client has to generate the evidence tokens and to send them periodically. It has to communicate with the URP client to get the current Session-ID. The Client also has to communicate with the marker software to know when the user starts to use a service. The marker’s task is to set the DSCP flag for the first time the user starts to use the service. The third interface is the communication to the metering software. It is responsible for the triggering of sending evidence tokens if the period is based on the amount of transfered data. The interaction between these four entities will be described in section 3 “Implementation”.

2.4 The Key Management

The Key Management is a general problem. This is the point the Trusted Third Party (TTP) may come into play. The goal of the Key Management is to give evidence that a public key belongs to a certain person. A certificate binds the public key and the information about the person with a signature. The signature is provided by a Certificate Authority (CA) acting as the TTP. This type of a TTP is called an offline TTP since the TTP doesn’t take part in the protocol. The disadvantages of the involvement of a TTP are the costs of getting a certificate for each user and the effort of creating the certificates.

A solution without a TTP is possible with some restrictions. Each customer has to generate the public/private key pair on its own and has to hand the public key over to the provider. The provider stores the public key in a database and is sure that the public key belongs to this customer. The problem arising is that the customer may claim the provider had generated the public/private key pair and the evidence tokens on their own. To hand out a certificate of the public key to the customer doesn’t solve the problem since the problem is the advantage of the customer over the provider. A solution is the use of a non-electronic evidence such as the customers signature over his public key as part of the contract.

2.5 Alternatives and Optimizations

This section discusses the alternatives and optimizations to the above presented solution. The advantages and disadvantages will be shown.

One of the alternatives is the integration of the Non-Repudiation security service into the URP and AAA protocol. The URP and AAA protocols are easily extendable by using new Type Length Value (TLV) triples and new Attribute Value Pairs (AVP) respectively. Easier Integration since both protocols already exist, less modification of the existing architecture and easier implementation are the main advantages of this design. Bad modularity, dependence upon used URP and AAA protocols and misusing of them are the disadvantages.

Another point bringing up alternatives is the question of what has to be signed and by whom. An alternative is that the client signs the AAA Request. The advantage of this is speedup since registration and evidence token movement will be done simultaneous. The evidence token can only be used to prove that the client has tried to register but not that he got access.
Packets signed by the Provider could be added optionally. Since the clients' devices aren't reliable it would be a bad idea to base the Non-Repudiation security service on packets signed by the Provider. Signing a response to a Reg Info or to a Service Info packet provides the client with the evidence that the access or service is granted. This is useful if the service isn't accessable although the client has sent the signed Service Info packet. To prove that the provider didn't grant the access to the service, the client has to prove that the access was denied, which is difficult to be proved.

If the period of sending Service Usage packets is variable it has also to be signed. For example if the period is constant within a session its value should be added to the Registration-Evidence-Token.

Since many evidence tokens will be created an optimization would be the use of a receipt after service usage. During the logout process the provider would send a receipt which the client would have to sign and return. The advantage of this optimization would be reducing of memory usage. The provider would have to store only the signed receipt and could remove the collected evidence tokens.

2.6 Attacks or Misbehaviour

Like all other security mechanism there could be security holes if the configuration is badly chosen. This section shows the important points which have to be treated carefully.

Attacks on the hash chain: There are two possible attacks on the hash chain. The first is the prediction of the random input r. If the provider is

able to predict r it could generate the Service Usage packets on its own. The other possible attack would be the precomputing of all possible hash chains. As soon as the provider has received a Service Info packet containing a main hash H_m, which is already known,the provider could generate the following Service Usage packets. To meet these problems an unpredictable random input is fundamental. Also important is the use of a good hash algorithm like SHA-1 or RIPEMD160.
A possible configuration could be a provider with 1'000'000 user which are 2 hours online a day. The period is set to 10 seconds and the number of hashes m is set to 30. The question is "how long does it take that all possible hash values are used once". Using SHA-1 or RIPEMD160 algorithms which produce 160 bit long hashes it would take $4 \cdot 10^{39}$ days.
In case of precomputing of all possible hash chains it would be necessary to have a storage of $2^{160} \cdot 20$ bytes $= 1.46 \cdot 10^{39}$ GB.

Attacks done by third parties: This type of attacks isn't one the Non-Repudiation protocol design has to concern of, since the link between the Client and Access Router is secured by IPSec's Authentication Header (AH). The IPSec AH is set up while registration. Though the Non-Repudiation Protocol protects against this sort of attacks. The Reg Info and the Service Info packet are signed by the client which are therefore protected against the attack since the public key is stored in the providers database and no key exchange is needed.

Attacks on the private key: Keeping the private key secret is the most important security measure. As soon as the private key is compromised the signature does not give evidence and the Non-Repudiation service isn't provided anymore.
The private key is stored on the customer's device. Well restricted access to the private key is fundamental.

Session-ID: The Session-ID is created during registration and has to be unique for each user. The signature of the Registration-Evidence-Token signs the NAI and the Session-ID. If the Session ID isn't unique for each user, the provider would be able to copy the signature. The effect isn't serious since the Registration-Evidence-Token doesn't contain any information about service consumption.

Lost or Missorder of Packets: The question is whether security holes arise if packets are lost or missordered. The packets contain a sequence number. The NAI, Session-ID and the sequence number identify the packet uniquely. If the Non-Repudiation Server received a duplicate it drops the packet without sending a NAK. In case of a packet which is received too

early the server drops it too, but sends a NAK message to the client. The provider may stop the service consumption as soon as a packet is lost or misordered or it may wait for a short time before stopping.
The customer may take an advantage of this short time. He could use the service without paying for that short time. The amount of this short time is a tradeof between usability and lost of evidence for that time.

3 Implementation

In the following each implemented entity will be described in detail. The used URP is the Client Authentication and Authorization Protocol (CAAP) which is the result of Christian Schlatter's diploma thesis "Development of a AAA System allowing controlled Access to IPv6 Intranets" [Sch02].
As shown in figure 3 the Non-Repudiation Server (non_rep_s), the Non-Repudiation Relay Agent (non_rep_a) and the Non-Repudiation Client (non_rep_c) had to be implemented.

3.1 CAAP and dia_dummy

A client (pep6_c), a network access server (pep6_s) and a dummy RADIUS Server were implemented by Schlatter. Since it was the task to use Diameter as AAA protocol a new Diameter server (dia_dummy) had to be written and the network access server code (pep6_dia) had to be extended with the Diameter protocol. On the clients' side, only minor changes were necessary. The adjusted client (caap_c) had to be extended with the sending of an IPC message as soon as the Session-ID received.

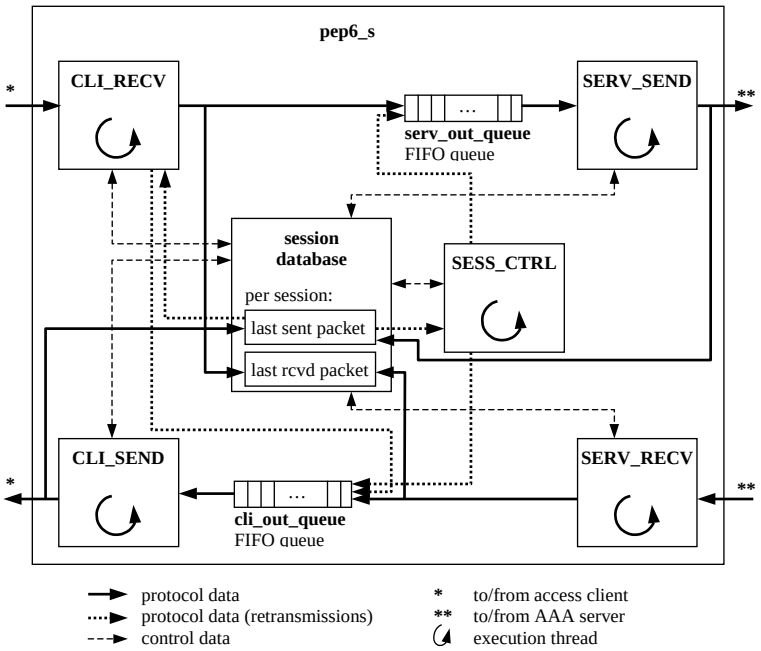

Figure 4: Structure of pep6_s [Sch02]

In figure 4 a diagram about the pep6_s is shown. The extension

with the Diameter protocol affected the two threads SERV_SEND and SERV_RECV. The AAA protocol used by Schlatter was RADIUS which is based on UDP. The Diameter protocol instead uses the TCP. To implement the Diameter protocol an Open Source library opendiameter [ODM] was used. A new Diameter command NAS_REQ was defined. The request and the response are composed by Attribute Value Pairs. Table 1 lists the AVPs and their data types. The AVPs written in square brackets are optional.

Table 1: AVP List of NAS_REQ command

Request	
-Session-ID	octet stream
-Origin-Host	octet stream
-Origin-Realm	octet stream
-Destination-Realm	octet stream
-Auth-Appl-ID	32 bit integer
-Chap-Challenge	octet stream
-Chap-ID	octet stream
-Chap-Response	octet stream
-NAI	octet stream
Response	
-Session-ID	octet stream
-Result-Code	32 bit integer
-[TSK]	$2 \cdot 128$ bit LSA key
-[Access-Parameter]	variable

The dia_dummy is a simple variant of the Diameter Server. Its task is to authenticate and authorize a single test user and to communicate with the Non-Repudiation Server. Figure 5 shows a diagram of the internal structure of the program. There's a receive and a send process for the communication with the Non-Repudiation Server. For each Access Router one process waits for NAS_REQ packets, handles them and sends the response packet. They also send IPC messages to the non_rep_send loop process containing information needed to compose MT Info packets.
The main process is responsible for listening on the AAA port and creates a new process as soon as an Access Router has connected to. The connections between the Diameter Server and each Access Router remains up until either part shuts down.
The non_rep_recv loop process waits for Diameter packets from the Non-

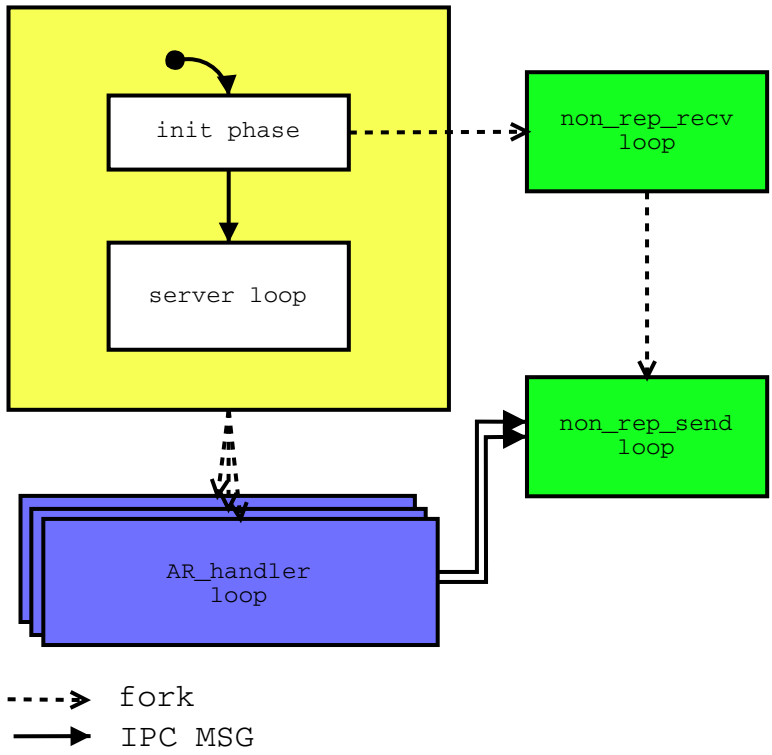

Figure 5: Structure of dia_dummy

Repudiation Server. These packets contain one of the commands listed in table 2. The table shows also the corresponding AVPs.

If the connection to the Non-Repudiation Server breaks down the non_rep_recv loop process tries to reconnect periodically. As soon as the reconnection was successful it will create a new non_rep_send loop process. This process waits for messages from any AR_handler loop process, generates and sends the MT Info packet to the Non-Repudiation Server. The AVPs of the MT Info packet are the following.

MT-Info	
-Reg-ID	grouped
-Session-ID	octet stream
-NAI	octet stream
-AR-Addr	octet stream

The generation of a new process after reconnection is necessary since the parent process and the client process do not share sockets created after forking.

Table 2: Diameter commands and their AVPs

Grant-Access	
-Reg-ID	grouped
-Session-ID	octet stream
-NAI	octet stream
Grant-Service	
-Reg-ID	grouped
-Session-ID	octet stream
-NAI	octet stream
-DSCP	32 bit integer
Stop-Service	
-Reg-ID	grouped
-Session-ID	octet stream
-NAI	octet stream
-DSCP	32 bit integer

3.2 Non-Repudiation Server Software non_rep_s

The Non-Repudiation Server has many tasks. First it's responsible to collect the evidence tokens and to verify them. It also has to communicate with a database to store them and to get the public key of each active user. Another task is to grant or stop service usage for each user. Taking decision when to stop service usage is a complex task.

Figure 6 shows the internal structure of the Non-Repudiation Server. The main process is responsible for verifying and storing of the evidence tokens. It maintains a dynamic list with an entry for each active user. The entry stores current information including the latest hash, latest sequence number and public key. The exact structure of the entry is presented in the appendix B.2.

The Interprocess Communication messages is used for transferring information from one process to the other. The main process waits for messages from any child process and handles them. A messages is stored in a queue until a process reads it. The queues are systemwide identified by a key. Each message contains a type which may be used as a priority number. Since one process can listen only on one queue, all processes who want to send a message to the main process have to use the queue with the fixed key NON_REP_SERVER_KEY. The types are defined as follows:

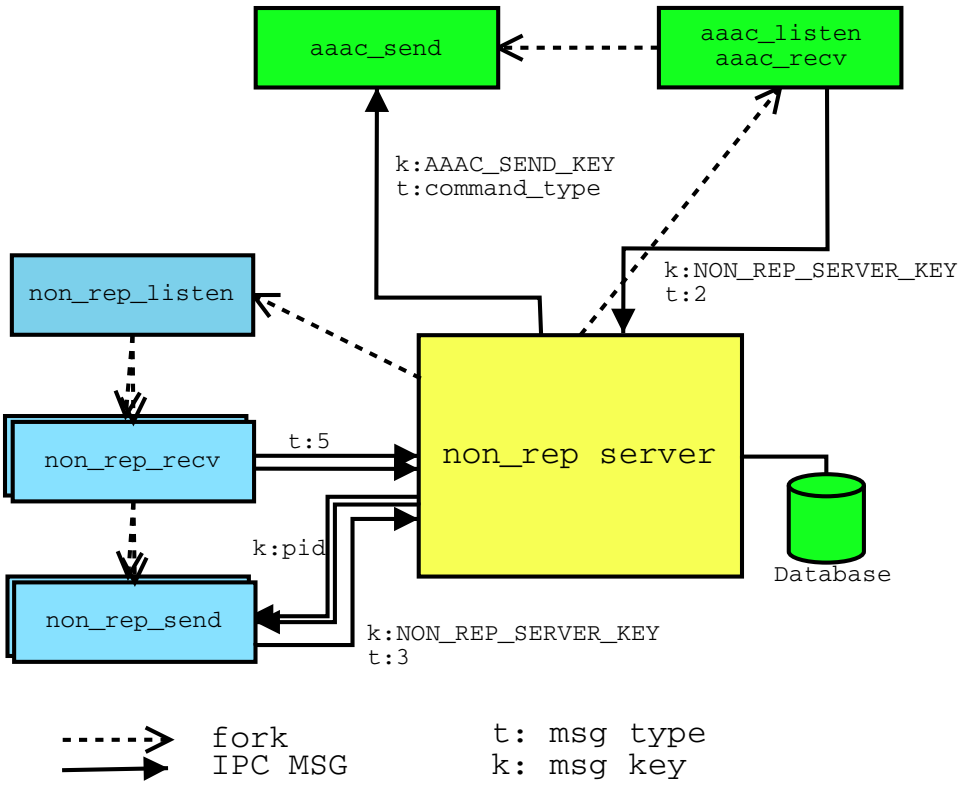

Figure 6: Structure of non_rep_s

- Type= 1: Error Message
- Type= 2: AAA-Comunication
- Type= 3: Relay Agent Registration
- Type= 5: Received Non-Repudiation Protocol Messages

The main process instead has to use different queues for sending messages to its child processes. The process aaac_send uses a fixed key. The non_rep_send processes use their process IDs (pids) as the keys.

The two processes aaac_send and aaac_recv are responsible for sending and receiving of Diameter packets to and from the AAAC Server. During the initial phase the aaac_recv process listens for the connection from the AAAC Server. As soon as the connection established the process creates the aaac_send process. If the connection breaks down the aaac_recv process kills the aaac_send process and listens for a new connection from the AAAC Server.

Since the server has to be able to communicate with each Non-Repudiation Relay Agent there's a receiving and a sending process for each Relay Agent. The non_rep_listen process waits for incoming connections from any Relay Agent. After a Relay Agent have connected to the Server two new non_rep_recv and non_rep_send processes are created. The non_rep_send process opens a message queue for receiving messages from the main process. As mentioned above the key of the message queue will be the process id. To let the main process know which key was used the non_rep_send process has to send a Relay Agent Registration message to the main process containing the key. The main process identifies each Relay Agent with their internal IPv6 address and uses a dynamic list with address-key pairs as map.
If the main process has to send a message like a NAK to a user, it searches for the current Access Router address the client is connected to and uses the map to get the corresponding key. The current Access Router address is stored in the per user entry of the list of active users. As soon as the client registers or performs a handover the AAAC Server sends the MT-Info message containing the address of the current Access Router. The sending of the MT-Info message is a new function to the current MobyDick AAAC Server. Besides, the current MobyDick AAAC Server is not aware of handover.

3.2.1 The Database

To store the evidence tokens a mysql database [MYS] is used. It has to store the Registration-Evidence-Token for each session a user begins and for each Registration-Evidence-Token one or more Service-Evidence-Tokens. For this there're three tables needed, one per user, one per Registration-Evidence-Token and one per Service-Evidence-Token.

Figure 7 shows the Entity Relationship(ER) diagram of the database structure. The table users stores the public key of each user. The User_id is the NAI and is used as the primary key of the table. The Certificate field has type of BLOB and contains the public key or the certificate in PEM [Ken93] format.

Apart from storing of the Registration-Evidence-Token the table Reg_evidence_tokens also stores additional information like date, time and the used cryptographic algorithms. The date and time are added by the Non-Repudiation Server and gives no evidence. It is added to make queries easier.
The algorithm identifiers may be provided by the Reg Info packet. If not, default values are taken. The Sig_alg identifier refers to the algorithm used

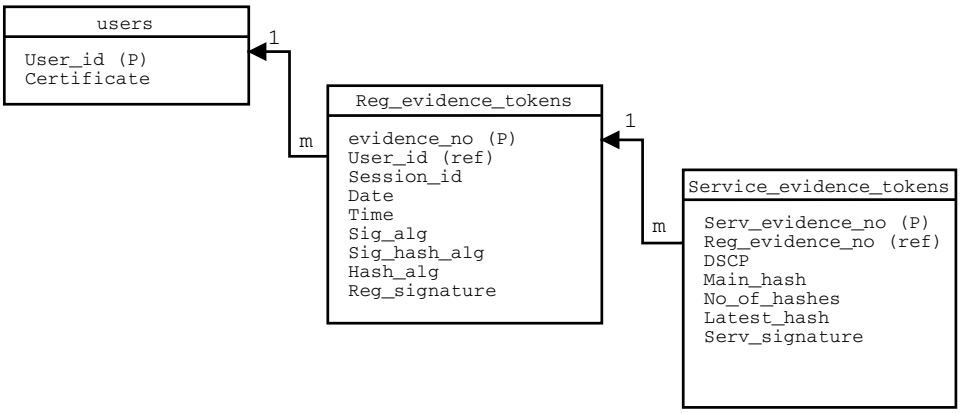

Figure 7: ER diagram of database

to sign the message digest, the Sig_hash_alg refers to the hash algorithm used to generate the message digest and the Hash_alg refers to the hash algorithm used by the hash chains.
The evidence_no is the automatic incremented primary key of the table. The User_id references the User_id of the users table in a 1 to m relationship. The Session_id and the Reg_signature are provided by the Reg Info packet.

The third table Service_evidence_tokens contains the evidence tokens for the service request and service usage. The Serv_evidence_no field constitutes the primary key of the table. It is unique within all tokens. The Reg_evidence_no references the evidence_no of the Reg_evidence_tokens table in a 1 to m relationship. The DSCP identifies the service type and is provided by the Service Info and Service Usage packet. The Main_hash is the H_m of the hash chain. The Service Usage packet provides always the newest H_n which will be stored as the Latest_hash. The No_of_hashes field is managed by the Non-Repudiation Server and counts the number $(m - n)$ of hashes received since the latest Main_hash.

3.3 Non-Repudiation Relay Agent Software non_rep_a

The Non-Repudiation Relay Agents task is to forward Non-Repudiation packets from the client to the server and vice versa. The communication between client and Relay Agent is based on UDP and the Relay Agent has to send an ACK back to the client for each received packet. On the other side the TCP is used between Relay Agent and server.

Since the communication between client and server is asynchronous,

the Relay Agent has to listen on both sides at the same time. This can be solved in a modular way by having more than one process. Figure 8 shows the structure of the relay agent software.

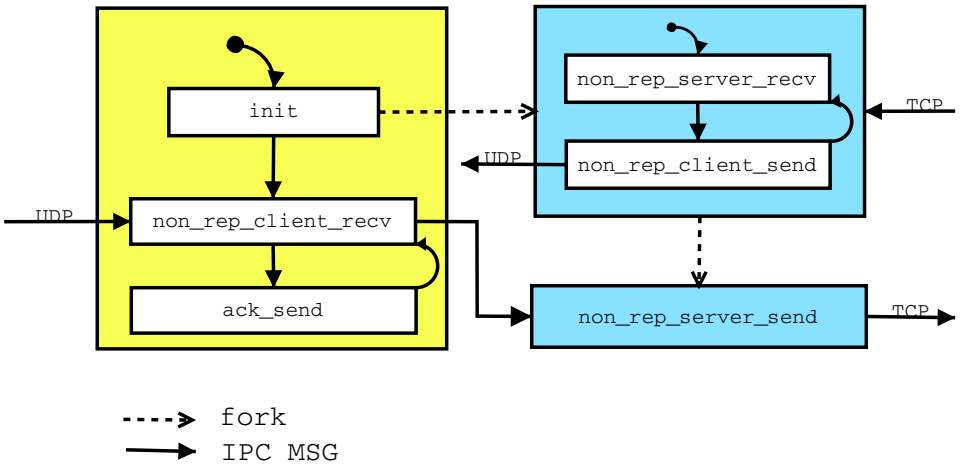

Figure 8: Structure of non_rep_a

After the initial phase the main process waits for UDP packets from any Client. As soon as a packet arrives it sends the acknowledgment packet ACK back and sends an IPC message to the non_rep_server_send process which is responsible for forwarding the packet through the TCP channel to the server. The third process is the non_rep_server_recv and waits for messages from the server. During the initial phase of this process it connects to the server and keeps the connection alive.
If the TCP connection to the server breaks down, the non_rep_server_recv process kills its child process non_rep_server_send and tries to reconnect to the server. As soon as the reconnection was successful it creates a new non_rep_server_send. This killing and recreation of the child is necessary to share the same TCP channel which is represented by the same socket.

3.4 Non-Repudiation Client Software non_rep_c

As described in chapter 2.2 “The Non-Repudiation Protocol” the client has to create the evidence tokens and to send them to the Network. As soon as the registration process is done the Registration-Evidence-Token has to be transmitted. The current implementation starts to send the Service Info and the Service Usage packets periodically after registration.

The Non-Repudiation Client consists of three parts: The main process, the Non-Repudiation Protocol processes and the trigger processes.

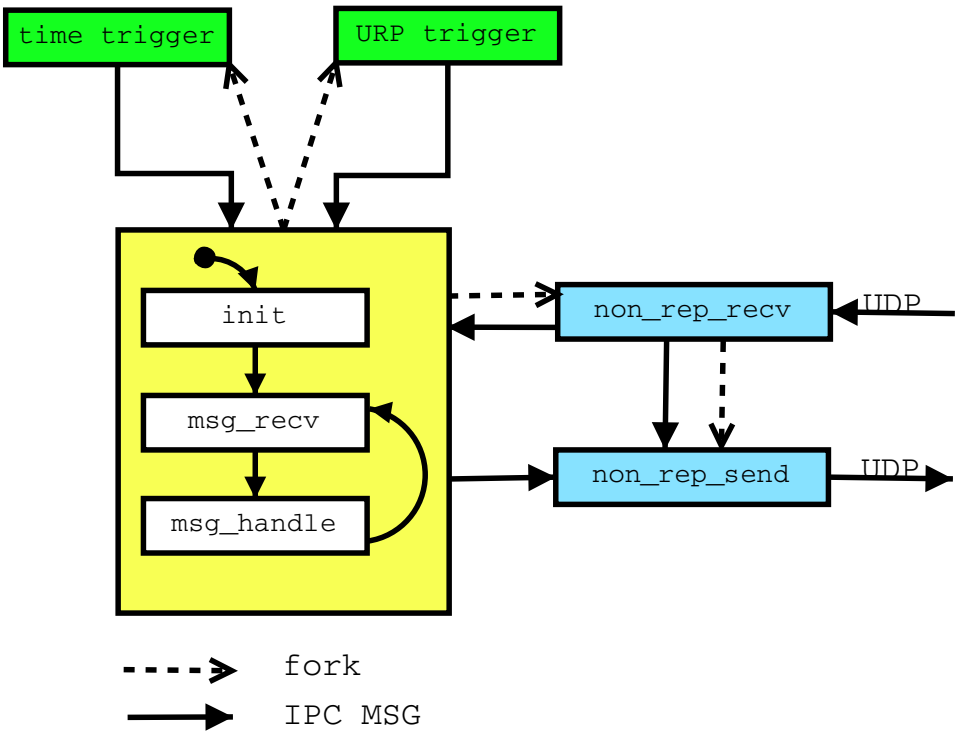

Figure 9: Structure of non_rep_c

Figure 9 depicts the structure of the client.

The Non-Repudiation Protocol processes are responsible for the communication with the network. The non_rep_recv process waits for any UDP packet from the current Access Router. There are two types of possible packets the client may receive from the network: An ACK or a NAK. If it is a NAK the packet will be send to the main process using a message. Otherwise if it is an ACK the process forwards it to the non_rep_send process. It is the task of the non_rep_send process to deliver the packets from the clients device to the current Access Router. It's responsible for resending the packets if no corresponding ACK was received within the timeout time. To accomplish this task the process uses signaling and a dynamic list of pending ACKs. The use of a specific signaling mechanism *sigqueue* allows to have more than one pending ACK. In this way the sending process does not have to wait for the acknowledgment of the previous packet.

The trigger processes have to inform the main process about certain events. The URP trigger process sends a message as soon as the URP client software sends the new Session-ID which means that the registration

was successful. The time trigger sends a messsage periodically to trigger the Service Info and Service Usage packets. The third trigger process, not implemented yet, is the meter trigger. It has to trigger the Service Info and the Service Usage packets if the period is based on the amount of transferred data. The reasons not to integrate the triggers into the main process and to use signals are modularity and better extendability.

After the initial phase the main process waits for IPC messages from any child process. The distinction of the messages is done by using different message types. Since the messages from the non_rep_recv process are always NAKs they got the highest priority and use therefore a low number as message type.
After receiving a message from the URP trigger the main process has to send the Reg Info packet. At this time it also creates the time trigger process and fills the hash chain for the first time. As soon as the Service Usage packet containing the latest hash of the hash chain was sent, it refills the chain with new hashes. This is done to not disturb the regularity of the periodic sending.

The formats of the IPC messages and the used structures are presented in appendix B.3.

4 Future Work

This section presents some open issues. It's important to mention that the provided implementation is a prototype of the Non-Repudiation Protocol.

Currently the Non-Repudiation Client stops its work as soon as a NAK received. To improve the usability its necessary to add a better NAK handling. Especially the receive of NAKs caused by misordered or lost packets wouldn't have always to cause a stop.

The sending of the Stop Service packet to the AAAC Server is currently an open issue. Stopping of the service usage as soon as a problem arise wouldn't be usable. Tests and measurements are necessary to get to know when stopping of service usage is needed. A possible solution is to wait a short time before stopping. If the problem is caused only by a delay of a packet there would be no reason to stop.
One may keep in mind that a too friendly configuration may arise security holes.

The current implementation doesn't provide a period, which is based on the amount of transferred data. To enable this feature the Non-Repudiation Client and the Relay Agent would have to interoperate with the meter software. The Non-Repudiation Server would have to collect the metering informations from the Relay Agents.

Another open issues is code optimizations. Since there are many users using services at the same time, the server has to be fast and may not use too much memory per active user. The server should be well scalable to the number of active user.

A Installation and Configuration

To use the software you need the source code of all components. The configurations are done by changing the defines in the header files. To compile the source code a simple

```
make
```

is needed. The Diameter Server, the CAAP Server and the Non-Repudiation Server need the opendiameter library [ODM] to link. In the following the configuration of each software package will be described. The section A.4 presents an example usage.

A.1 The CAAP software

The CAAP is implemented by the caap_c and the pep6_dia software packages. Both are derived from the software provided by Christian Schlatter. Appendix B of the report of Schlatters diploma thesis [Sch02] presents the installation and usage of the software in detail.
Before using pep6_dia it is necessary to insert the pep6_kmod module into the kernel. Starting the software with the help option `-h` shows the correct syntax.

The key to get the message queue of the Non-Repudiation Client may be changed in the caap_c code. The key is defined in the pep6_c.h header file as `URP_NRC_MSG_KEY`.

There are two port numbers to be set for the pep6_dia software. First the port the software is listening for CAAP Clients. This port number is defined in pep6_s.c as `SERVER_PORT`. The second port number to be set is the port the Diameter Server is listening on. The define `RAD_S_PORT` sets this port number.

A.2 The simple Diameter Server dia_dummy

To configure the simple Diameter Server you may change some define values set in the dummy_s.h header file. Table 3 lists the defines, their default value and their meaning.

The commandline usage is as follows:

```
./dummy_s -q cli_secret -u cli_name -s rad_secret
```

Table 3: Defines of dummy_s.h

Define	Default	Description
DEBUG	1	Enable Debug output
SERVER_PORT	1812	AAA Protocol Port
NON_REP_SERVER_PORT	1814	Port of the Non-Repudiation Server
TIMEOUT	10	Seconds waiting before reconnecting
NO_OF_RETRIES	10	Maximal number of reconnection tries
NON_REP_SERVER_ADDR	none	IPv6 address of Non-Repudiation Server

This allows the access for one client with the cli_secret password. The rad_secret must be the same as the one used with the CAAP Server.

A.3 The Non-Repudiation Software

The Non-Repudiation Server non_rep_s: The header File non_rep_structs.h contains defines which may be changed. The define NON_REP_SERVER_PORT sets the port number the server is waiting for connection from the AAAC Server. The second port the server is listening on is the NON_REP_PORT. The Non-Repudiation Relay Agents have to connect to this second port.
The define MT_IPV6_ADDR sets the IPv6 address of the client's Mobile Terminal. This is needed to send a NAK to the client.
The three defines DEF_SIG_ALG DEF_SIG_HASH_ALG DEF_HASH_ALG set the default cryptographic algorithms to be used for verifying the signatures, creating the message digests and verifying the hash chains respectively.

To start the server type

```
./non_rep_s
```

The nrdbadmin software is a tool to manage the Non-Repudiation database. The usage is:

```
./nrdbadmin command command_args
```

At the moment only the add command is implemented. It adds a new user with the given public key file to the database. The key file has to be in the PEM format. The usage of the add command is:

```
./nrdbadmin add username pubkeyfile
```

The file DB_create.sql can be used to create the database and to add the necessary tables. To do this use the following command:

```
mysql -hDBhost -uusername -ppassword < DB_create.sql
```

The Non-Repudiation Relay Agent non_rep_a: To configure the Relay Agent the defines in the header file non_rep_a.h have to be changed. The define `NON_REP_PORT` and `NON_REP_SERVER_ADDR` sets the port number and the IPv6 address the Non-Repudiation Server is listening on.
The two defines `TIMEOUT` and `NO_OF_RETRIES` define the behaviour after the connection to the Non-Repudiation Server breaks down. To start the server type

```
./non_rep_s
```

The Non-Repudiation Client non_rep_c: Table 4 lists all defines which may to be changed. The defines are located in the non_rep_c.h header file.

Table 4: Defines of non_rep_c.h

Define	Default	Description
`DEBUG`	1	Enable Debug output
`NON_REP_PORT`	1815	Port the Relay Agent is listening on
`NON_REP_AGENT_ADDR`	none	IPv6 address of Relay Agent
`TIMEOUT`	10	Seconds waiting before reconnecting
`NO_OF_RETRIES`	10	Maximal number of reconnection tries
`PRIVKEY_FILE`	”privkey.pem”	Location of private key file
`TRIGGER_TIME`	10	Period of sending evidence tokens
`NO_OF_HASHES`	10	Number of hashes per hash chain
`RANDOM_SIZE`	16	Size(words) of the random input r
`URP_NRC_MSG_KEY`	75	Key to URP message queue

The define `URP_NRC_MSG_KEY` must have the same value as the define `URP_NRC_MSG_KEY` of the caap_c client.

The software keygen, part of the non_rep_c package, provides the generation of the public and private key files. To create the files pubkey.pem and privkey.pem use:

```
./keygen
```

A.4 Example configuration

This section presents an example configuration. As you can see in figure 10 the configuration consists of three computers. On the MT the CAAP Client and the Non-Repudiation Client are installed. The ARC performs the Access Router and has the pep6_dia and the non_rep_a softwares installed. The third computer AAAC is used as the AAAC Server and the Non-Repudiation Server. A mysql server on the ARC is used to store the Non-Repudiation database.

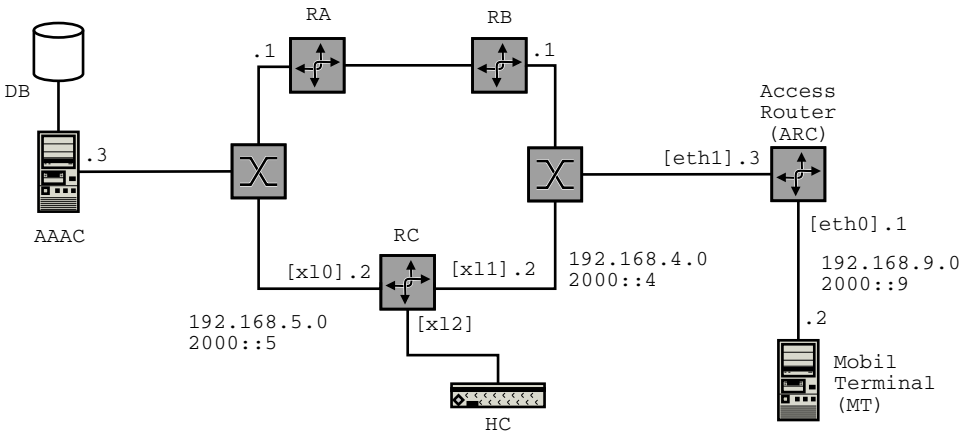

Figure 10: MobyDick example Network

All of the hosts use a RedHat Linux 7.2 with a patched standart kernel 2.4.16 installed. The patch [SWA] is needed to run both the FreeS/WAN and Mobile IP as modules.

To run it all start the Non-Repudiation Server and the AAAC Server on the AAAC machine:

```
./non_rep_s
./dummy_s ...
```

After it start the Relay Agent and the CAAP Server on the ARC machine:

```
./non_rep_a
./pep6_dia ...
```

At last start the Non-Repudiation Client and the CAAP Client:

```
./non_rep_c
./caap_c ...
```

As soon as the CAAP Client is started the client performs a Registration and sends the Reg Info packet if the client got authorized. After Registration the client sends the Service Info and Service Usage packets with DSCP set to 1 periodically.

To stop it `CTRL-C` had to be hit for each entity in reverse order they were started. The reverse order isn't mandatory but prevents the servers to block the ports for the TCP Time_Wait time.

B Structs and Protocol Formats

This appendix lists the most important structs and the formats of the Non-Repudiation Protocol packets. There're two types of structs important to list. The structs used to transfer information using IPC messages and the structs to store current information in dynamic lists.

B.1 Structs used by Messages

The reg_id struct is used by the most messages.

```
typedef struct reg_id {
    char session_id[MAX_SESSID_SIZE];
    char nai[MAX_NAI_SIZE];
    int  naisize;
    int  sessidsize;
} reg_id_t;
```

The mt_info_msg is used by the dia_dummy software between the AR_handler loop process and the non_rep_send loop process.

```
typedef struct mt_info_msg {
    long type; // Must be AAA_RECV_TYPE
    reg_id_t  reg_id;
    char addr[INET6_ADDRSTRLEN];
    int addrlen;
} mt_info_msg_t;
```

The Non-Repudiation Server uses three structs: The msgbuf to send Non-Repudiation packets to the non_rep_send process, the reg_relay to register a non_rep_send process with the main process and the aaac_send_msg to transfer information from the main process to the aaac_send process.

```
typedef struct msgbuf {
    long type;
    char text[MAX_MSG_DATA];
} msgbuf_t;
```

```
typedef struct reg_relay{
    long type;              // Must be REG_RELAY_TYPE
    struct in6_addr ar_addr;// IPv6 address of Relay Agent used as identifier
    int pid;                // process id of send process used as message key
} reg_relay_t;
```

```
typedef struct aaac_send_msg {
    long type; // GRANT_ACCESS | GRANT_SERVICE | STOP_SERVICE
    reg_id_t  reg_id;
    int dscp;        // not used if type = GRANT_ACCESS
} aaac_send_msg_t;
```

The Non-Repudiation Client uses the msgbuf struct to transfer Non-Repudiation packets between the processes. The Trigger processes use the timetriggermsg and the urptriggermsg to trigger the main process. The timetriggermsg doesn't contain any information. The urptriggermsg describes the messages exchanged with the CAAP client.

```
typedef struct timetriggermsg {
    long type;
    char *text;
} timetriggermsg_t;
```

```
typedef struct urptriggermsg {
    long type;
    int  sessid_len;
    int  nai_len;
    char sessid;     // first byte of session id, nai is appended to the session id
} urptriggermsg_t;
```

B.2 Structs used by dynamic Lists

The Non-Repudiation Server manages two dynamic lists. The first stores the corresponding message key for each Access Router. The structure of an entity is defined by ar_key.

```
typedef struct ar_key {
    struct in6_addr ar_addr;
    int key;
    ar_key *next;              //pointer to the next element in list
} ar_key_t;
```

The second list stores current information about each registered user. The nai_info defines the structure of an entity.
The Non-Repudiation Client manages one list. The list contains senditem entities and is used by the non_rep_send process to manage the sent but not yet acknowledged packets.

```
typedef struct nai_info {
    // stores the nai - ar_addr relation and other nai information
    char session_id[SESSION_ID_SIZE];
    char nai[MAX_NAI_SIZE];
    int  nai_size;
    struct in6_addr ar_addr;
    unsigned int latest_seqno;
    uint8_t  sig_alg;
    uint8_t  sig_hash_alg;
    uint8_t  hash_alg;
    char *latest_hash;
    nai_info *next;              // pointer to the next element in list
} nai_info_t;


typedef struct senditem {
    unsigned int seqno;
    int no_of_timeouts;
    senditem *next;
    int msglen;
    char msg[MAX_MSG_LEN];
}  senditem_t;
```

B.3 Protocol Formats

This part of the appendix presents the formats of the Non-Repudiation packets. There're three types of packets: The main packet type to send the Reg Info, Service Info and Service Usage packets, the ACK packet type to send the acknowledge message from the Access Router to the Mobile Terminal and the NAK packet type which may be send by the Non-Repudiation Server to the Mobile Terminal through the Access Router. In the following each packet type will be described in detail.

The main packet type: Figure 11 shows the format of this type. It consists of one header and one or more Type Length Value triples (TLVs). Table 5 lists all current defined TLVs.

The header consists of a 4 bit msg type field, a 4 bit reserved field, a 16 bit totallength field containing the length of the packet including the header, a 32 bit sequence number field, a Session-ID size field, a NAI size field and two fields of variable length containing the Session-ID and the NAI. A TLV consists of a 8 bit type field, a 8 bit length field containing the length of the value field in bytes and a value field of variable length. Figure 12 presents the TLV format.

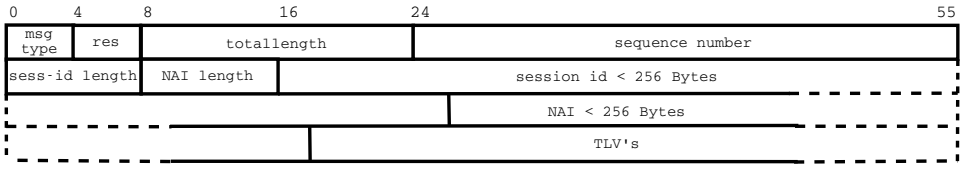

Figure 11: Header Format of the Main Packet Type

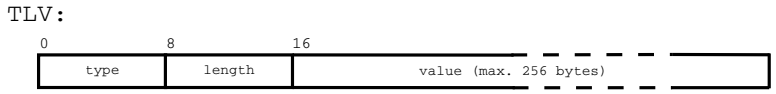

Figure 12: TLV Format

Table 5: List of defined TLVs

Type	Length	Value
1 - Signature	Algorithm dependent	Signature
2 - HASH	Algorithm dependent	Hash
3 - DSCP	1 byte	DSCP
4 - SIG_ALG	1	Algorithm code
5 - SIG_HASH_ALG	1	Algorithm code
6 - HASH_ALG	1	Algorithm code

The ACK packet type: The ACK is kept small and consists of a 4 bit msg type field, a 4 bit reserved field and a sequence number field. The sequence number contains the sequence number of the packet the ACK acknowledges. Figure 13 presents the format of the ACK packet.

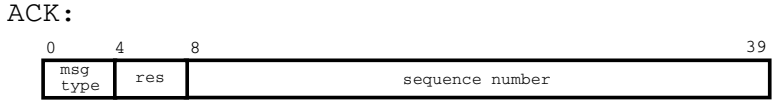

Figure 13: ACK Packet Format

The NAK packet type: The NAK packet will be send by the Non-Repudiation Server and has to be forwarded by the Relay Agent. Since the Relay Agent isn't able to get the IPv6 address out of the NAI, the NAK packet contains the current IPv6 address of the client.
The format of the packet is shown in figure 14. The NAK packet consists of

a 4 bit message type field, a 4 bit reserved field, a 32 bit sequence number field containing the sequence number of the packet which caused the NAK, a 8 bit NAK code field, a 32 bit NAK info field and a 128 bit IPv6 field. Table 6 lists all current defined NAK codes and the corresponding content of the NAK info field.

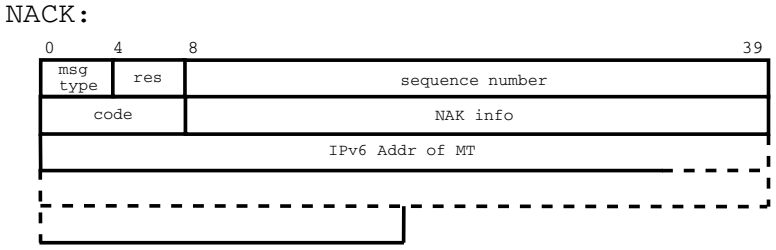

Figure 14: NAK Packet Format

Table 6: List of defined NAK codes

Code	NAK Info Field
01 - BAD_NAI	empty
02 - NO_KEY	empty
03 - NO_REG	empty
04 - BAD_SESSID_LEN	Correct Session ID length
05 - BAD_SESSID	empty
06 - EARLY_PACKET	latest hash number
07 - MISSING_TLV	empty
08 - BAD_SIGNATURE	empty
09 - UNSUPPORTED_ALG	Type of algorithm (1=sig, 2=sig_hash, 3=hash)
10 - UNSUPPORTED_MSG_TYPE	empty
11 - NAI_NOT_REGISTERED	empty
12 - BAD_HASH	empty

References

[ISO97] ISO/IEC 13888-1. Information Technology - Security techniques - Non-repudiation - Part 1: General. ISO/IEC, 1997.

[IST01] IST-2000-25394 Project Moby Dick. Moby Dick Framework Specification. Public Deliverable (except Appendix A and B), D0101, November 2001.

[IST02a] IST-2000-25394 Project Moby Dick. AAAC Design. Public Deliverable, D0401, January 2002.

[IST02b] IST-2000-25394 Project Moby Dick. Initial Design and Specification of a Moby Dick Mobility Architecture. Internal circulation within the project, D0301, April 2002.

[IST02c] IST-2000-25394 Project Moby Dick. Initial Design and Specification of a Moby Dick QoS Architecture. Internal circulation within the project, D0201, April 2002.

[IST02d] IST-2000-25394 Project Moby Dick. Moby Dick Signaling Flow Specification and Implementation. Specification with respect to Mobility, QoS and AAA. Task Internal Report, October 2002.

[JD96] J.Zhou and D.Gollmann. A fair non-repudiation protocol. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.

[JD97] J.Zhou and D.Gollmann. Evidence and non-repudiation. *Journal of Network and Computer Applications*, 20(3):267–281, July 1997.

[Ken93] S. Kent. Privacy Enhancement for Internet Electronic Mail. Part II: Certificate-Based Key Management. RFC 1422, February 1993.

[MDP] The official MobyDick project's web page. `http://www.ist-mobydick.org`.

[MDT] MobyDick project's web page maintained by ETH Zürich. `http://www.tik.ee.ethz.ch/~mobydick`.

[MYS] MySQL - Open Source implementation of a relational Database. `http://www.mysql.org`.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

[ODM] opendiameter - Open Source implementation of Diameter. `http://www.opendiameter.org`.

[P.C02] P.Calhoun. Diameter Base Protocol. IETF work on progress, draft-ietf-aaa-diameter-15.txt, October 2002.

[Sch02] Christian Schlatter. Development of a AAA System allowing controlled Access to IPv6 Intranets. Diploma Thesis, D-ITET and D-INFK, ETH, Zürich, Feb 2002.

[SSL] OpenSSL - Open Source implementation of the SSL and TLS protocol. `http://www.openssl.org`.

[SWA] SWAMP - Secure Wide Area Mobility Package. `http://www.cwc.nus.edu.sg/~parijat/swamp/`.