TAMPEREEN TEKNILLINEN YLIOPISTO

SANTERI SILTALA

NON-REPUDIATION SERVICE IMPLEMENTATION USING HOST IDENTITY PROTOCOL

Master of Science Thesis

Examiners: Prof. Jarmo Harju
D.Sc Seppo Heikkinen

Examiners and topic approved in the Faculty of Computing and Electrical Engineering Council meeting on 7.3.2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications
SILTALA, SANTERI: Non-Repudiation Service Implementation Using Host Identity Protocol
Master of Science Thesis, 62 pages + 1 appendix page
May 2012
Major: Communications Networks and Protocols
Examiners: Professor Jarmo Harju, D.Sc Seppo Heikkinen
Keywords: Non-repudiation, Host Identity Protocol, service usage, accounting, RADIUS

New types of service usages emerge every day in the Internet. Service usage could be Wireless Local Area Network (WLAN) usage or watching a streamed movie. Many of these services are commercial, so payment is often involved in the service usage, which increases the risk of fraud or other misbehaviour in the interaction. To enhance the security of both service providers and service users, improvements are needed to the existing procedures.

The non-repudiable service usage procedure was developed as part of the TIVIT Future Internet SHOK -project. In this model, the service user and the service provider are bound to the actual service usage with certificates. The charging of the service usage is done using hash chains which are bound to the certificates. Now the service user pays only for the service he or she gets. Time or traffic based charging scheme can be used in the service usage. Evidence is gathered from the service usage to help solve possible conflicts afterwards.

An actual implementation based on this model was made using Host Identity Protocol for Linux and RADIUS protocol. RADIUS protocol was used to gather the created evidence of the service usage. The implementation was developed for Linux using C-language. The goal of the implementation was to evaluate the concept in actual use. Performance of the implementation was measured with various real use scenarios to evaluate the feasibility of the implementation. Results indicated that the performance of the model is sufficient to serve several simultaneous users. However, the architecture of Host Identity Protocol for Linux caused some performance issues in the implementation.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
SILTALA, SANTERI: Kiistämättömän palvelutarjonnan toteuttaminen HIP-protokollan avulla
Diplomityö, 62 sivua + 1 liitesivu
Toukokuu 2012
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastajat: professori Jarmo Harju, tekniikan tohtori Seppo Heikkinen
Avainsanat: Kiistämättömyys, HIP-protokolla, RADIUS-protokolla, palvelun käyttö, tiliöintitiedot

Erilaisten palveluiden käyttö Internetissä kasvaa koko ajan. Tämä voi pitää sisällään esimerkiksi langattoman lähiverkon käyttöä tai virtautetun elokuvan katsomista. Usein palveluiden käyttö on kaupallista, joten käyttämiseen liittyy myös maksutapahtumia, mikä lisää huijatuksi joutumisen riskiä. Palveluiden käyttöä tulee kehittää, jotta voidaan parantaa palveluiden käyttäjien ja tarjoajien turvallisuutta.

Tämän diplomityön taustalla olleessa TIVIT Future Internet SHOK -projektissa on kehitetty toimintatapaa, jolla palvelun käytöstä tehdään kiistämätöntä. Tässä toimintatavassa palvelun tarjoaja ja käyttäjä sidotaan kiistämättömästi palvelun käyttöön varmenteiden avulla. Käyttäjä maksaa vain siitä palvelusta, jota oikeasti saa. Maksaminen tehdään käytön edetessä varmenteisiin sidottujen hajautusketjujen avulla. Käytön laskutus voidaan tehdä esimerkiksi aika- tai käyttömääräperusteisesti. Palvelun käytöstä kerätään todisteet, joita voidaan käyttää mahdollisten väärinkäyttötilanteiden selvittämisessä.

Työssä kehitettiin Host Identity ja RADIUS-protokolliin pohjautuva toteutus kiistämättömästä palvelunkäytöstä. RADIUS-protokollaa käytettiin apuna kerättyjen todisteiden säilöntään. Tavoitteena oli tehdä prototyyppiohjelmisto, jolla voisi tutkia kehitetyn konseptin soveltuvuutta käytäntöön. Toteutus tehtiin Linux-käyttöjärjestelmälle käyttäen C-kieltä. Toteutuksen suorituskykyä erilaisissa tilanteissa mitattiin, jotta voitiin arvioida sen soveltuvuutta jokapäiväiseen käyttöön, erityisesti palveluiden tarjoajan näkökulmasta. Saatujen tulosten perusteella voidaan todeta, että toimintatapa soveltuu muutamille yhtäaikaisille käyttäjille. Mittauksissa kuitenkin havaittiin, että pohjana käytetty Linux-pohjainen HIP-protokollan toteutus sisältää suorituskykyongelmia, jotka saattavat aiheuttaa koko toteutuksen hidastumista usean yhtäaikaisen käyttäjän tapauksessa.

FOREWORD

This Master of Science thesis was done in the Department of Communication Engineering (DCE) at the Tampere University of Technology (TUT). This work was part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). This program is funded by TEKES.

I want to thank my supervisors, Professor Jarmo Harju and D.Sc Seppo Heikkinen. They made it possible for me to participate in this interesting research project and supported me during the whole thesis process. I also want to thank my co-workers D.Sc Jani Peltotalo and M.Sc. Aleksi Suhonen who gave help and great thoughts throughout the thesis process.

Last but not least I would also like thank by beloved girlfriend for the patience during the writing process. I want also to thank my parents who provided support during the whole study time.

On 11.4.2012, in Tampere, Finland.

Santeri Siltala

TABLE OF CONTENTS

LIST OF ACRONYMS

AAA	Authentication, Authorization and Accounting
API	Application Programming Interface
BEET	Bound End-to-End Tunnel
CA	Certificate Authority
D-H	Diffie-Hellman
DOS	Denial of Service
DNS	Domain Name System
DSA	Digital Signature Algorithm
ESP	Encapsulating Security Payload
HADB	Host Association Database
HI	Host Identifier
HIP	Host Identity Protocol
HIPL	Host Identity Protocol for Linux
HIT	Host Identity Tag
HMAC	Hashed Message Authentication Code
IKE	Internet Key Exchange Protocol
IP	Internet Protocol
LSI	Local Scope Identifier
MAC	Medium Access Control
MitM	Man in the Middle
NAS	Network Access Server
NAT	Network Address Translation
NoRSU	Non-repudiable Service Usage
NRD	Non-repudiation of Delivery
NRO	Non-repudiation of Origin
NRR	Non-repudiation of Receipt
NRS	Non-repudiation of Submission
ORCHID	Overlay Routable Cryptographic Hash Identifier
PAP	Password Authentication Protocol
RADIUS	Remote Dial In User Service
RSA	Rivest, Shamir, Adleman
SPKI	Simple Public Key Infrastructure
TCP	Transmission Control Protocol
TTP	Trusted Third Party
UDP	User Datagram Protocol
VSA	Vendor Specific Attribute
WLAN	Wireless Local Area Network

1 INTRODUCTION

More and more transactions are made in the computer networks every day. The variety of services of a different kind also increases. The most of the time users have to interact with unknown actors, and the interaction is based solely on trust. Even the business between two operators is often based on trust. This increases the demand for methods of interacting reliably between players.

In today's global networks it is nearly impossible to get a bulletproof protection against frauds and misbehaving individuals and service providers. Many popular video on demand or music on demand services do not provide any kind of protection for the paying customers. In case of fraud or malfunction, the risk for the customer to lose his or her payment is considerable. However, this risk of being mistreated can be greatly reduced with some simple improvements to the interaction between the customer and the service provider. Now more and more customers have awakened to demand more reliability to the service usage which takes place in the Internet. Offering secure service usage can be a competitive advantage to the operators in the future. Still, there have not been any widespread implementations to offer this kind of service usage.

Non-repudiation aims to give theoretical tools to satisfy the aforementioned needs. Even though non-repudiation is a well-researched topic, it still lacks actual implementations. The secure service usage requires undeniability of actions to both, the operator and the customer. When these actors form a contract, for example in some kind of service usage which includes payments, either of the sides must not be able to cheat easily in any way. It is possible to minimize the risk in the payment transaction by splitting the payment into multiple smaller parts in a pay-as-you-get-service manner. Still, there needs to be a secure way to transmit these smaller payment units to the service provider.

The purpose of this implementation is to provide a technical prototype solution to these problems. The main focus of the implementation is to research and test how well these theoretical procedures can be fitted into an actual application and whether the performance is satisfactory to a larger scale operation. One target is to evaluate how feasible this solution would be in an actual operator use.

The structure of the thesis is as follows. The second chapter presents the main concepts of the non-repudiation theory. It describes the main types and subtypes of non-

repudiation and elements and players which are needed in a typical non-repudiable transaction. The third chapter contains an introduction to the technologies used in the prototype implementation. The chosen technologies are analyzed and compared to other existing similar technologies. There is also a brief introduction to the authentication, authorization and accounting protocols. The fourth chapter contains the functional and the non-functional requirements which were set for the design. The fifth chapter contains an introduction to the design principles, module descriptions and the architecture of the implementation. The implementation includes several operation modes which are explained in detail. The sixth chapter contains actual implementation specific issues such as the data structures, parameter types and sequence diagrams. The interaction between different components is presented in several use cases. The seventh chapter consists of testing and analysis. The test setup and environment is introduced, as well as the execution of the tests. The test results are analyzed and fulfillment of the functional and the non-functional requirements is examined. The eight chapter is about future work and how this implementation could be developed further. The ninth chapter concludes the work.

2 BACKGROUND CONCEPTS

Non-repudation theory is based on some key elements, which are required to achieve the reliable non-repudiation service. There is some crucial evidence which must be collected and in most of the cases help of some external party is needed. Non-repudiation can be used in various applications such as electronic commerce or non-repudiable contracts.

2.1 Purpose of Non-Repudiation

Non-repudiation is a mechanism to bind all the participating entities to the committed transaction. This means that none of the parties can deny afterwards their involvement in the transaction. In a typical situation a dishonest entity could deny its participation or claim that some evidence related to the transaction, such as the signature, is forged.

Non-repudiation services are mainly needed to support business transactions which take place in an insecure environment, such as the Internet. In the electronic transactions it is much harder to ensure involving parties' identities than in the traditional transactions. Also the probability of fraud is much greater. Especially, when there is payment involved in the transactions, it is crucial to bind the entities to the transaction.

In the traditional cases signature is typically used to ensure non-repudiation. One's signature in a contract proves that he or she has accepted the terms presented and is willing to follow them. Typically all of the involved parties get a copy of the signed agreement as evidence. Confirmation of entities' identities is also easy to verify by using, e.g., the passport, which includes one's signature and photograph and is considered legally as a strong evidence of one's identity. Still, there is a possibility of fraud, so the most important agreements require the presence of some trusted third party such as a notary.

Typically, the following entities are involved in the non-repudiable transaction which provides evidence [1]:

- Non-repudiation of origin (NRO). The origin must provide evidence that it really has sent the message. The proof of origin is required to prevent cases where a dishonest entity may afterwards try to deny being involved in the communication. The focus of the evidence is to achieve non-repudiation instead of just providing evidence that the message has been sent.

- Non-repudiation of receipt (NRR). The receiver must provide evidence that it really has received the message. Proof of receipt is required to prevent cases similar to the previous case.
- Non-repudiation of delivery (NRD). The transmission entity must provide evidence that it has received the message from the origin to be delivered to the recipient.
- Non-repudiation of submission (NRS). The Transmission entity must provide evidence to the origin that it has delivered the message to recipient.

Non-repudiation aims to solve these introduced problems in an electronic environment. This requires gathering certain evidence of the transaction between the entities. A non-repudiable message delivery process is described in Figure 2.1 [2]. The sender creates the message and is responsible for providing the proof of origin. The user agent handles the message to the actual delivery instance. Together the sender and the user agent form the origin.

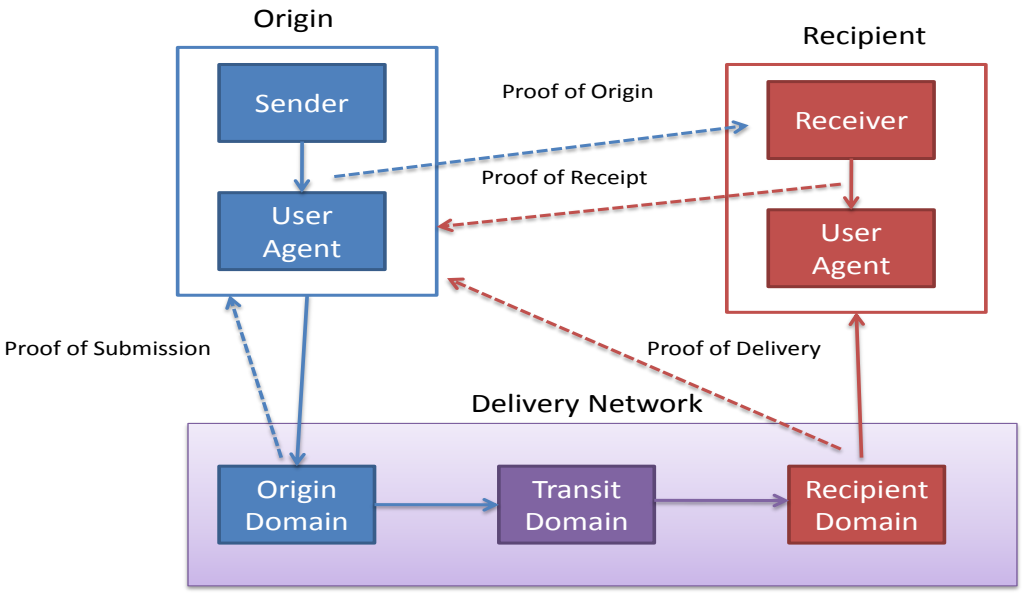

Figure 2.1. *Typical evidence in non-repudiation.[2]*

The delivery network consists of several domains through which the messages are delivered. The first transmission entity, which receives the message from the user agent, is called the origin domain and must provide the proof of submission. The last transmission entity in the delivery network provides the proof of delivery to the origin. The receiver and the user agent form the recipient side must provide the proof of receipt.

2.2 Fairness

An important requirement for non-repudiation is fairness. In the business cases it is important to try to balance the risk of being defrauded or mistreated. Typically, one of the

parties is slightly in a weaker position in the risk sharing, but non-repudiation aims to provide a fair interaction. In [3] the following classification for the fairness levels and requirements is presented.

In weak fairness, the fairness of the exchange is not guaranteed, but the wronged party will get evidence that the exchange was not fair, while it may have lost the item or the message used in the exchange. In an example case a dishonest shopkeeper might not send the item after having received the payment from the customer. However, the customer has proof from the bank that he or she has paid the product. Gathered evidence can be processed in the court or by other authority afterwards. Protocols based on probabilistic fairness provide the second weakest level of fairness. Execution of the protocol provides high level probability for fair exchange while it still leaves the possibility that unfair exchange can occur. The motivation for using this kind of fairness is that it does not require the aid of Trusted Third Party (TTP) in the exchange. In strong fairness, if an exchange is finished successfully, both sides must get the appropriate evidence and items. If an error occurs in the execution, either side must not get anything. True fair exchange must provide the properties of the strong fairness described earlier and the evidence created during the communication must not depend on how the protocol was executed. So the use of the TTP must not affect the execution of protocol or creation of the evidence to achieve true fairness.

When true fairness is not feasible, the other solution is to try to minimize one's loss in the situations of misbehavior. Non-repudiation is not a requirement for fairness but evidence generated by the non-repudiation actions may be used afterwards to prove one's misbehaving actions [4][5]. NRO and NRR mentioned in section 2.1 are the minimal evidence needed to ensure the fair transaction.

2.3 Trusted Third Party

Trusted Third Party is an entity which is used to carry out transactions between entities which do not trust each other. Generally, TTP is considered as the reliable entity for both entities. If either finds lack of trust towards the TTP, it typically compromises all non-repudiation and fairness procedures which rely on the use of the TTP. Because of this issue, there is always possibility of fraud, since it is possible that even an extremely trustworthy TTP begins to misbehave at some point.

Use of the TTP can be divided into three main categories, online, inline and offline TTP. An online TTP is available all the time during the transaction. It may supervise the procedure, collect evidence or it may act as a verifier for the certificates at the beginning of the transaction. The most important thing is that both of the transaction entities may rely on the TTP at any point of the transaction. If the possibility to reach the TTP is lost during the transaction, the procedure may be compromised.

Inline TTP acts as a broker between the communicating entities. This requires that the both entities must trust the same TTP and TTP must have all communicating entities' public keys pairwise or use symmetric keys. Mechanism causes privacy issues as the messages and transactions go through the TTP and TTP is able to examine the content of the messages. One solution is that TTP can decrypt only part of the message but this adds demand to bind the originator to the message. Inline TTP may also become a bottleneck in the traffic.

Another possible situation is that TTP is partly or most of the time of the transaction offline. In many cases the help of TTP is only needed when some problems occur during the interaction. The problem could be a network error or misbehavior of an entity. In some cases, both entities deliver collected evidences to the TTP which makes decisions based on those and helps the transaction to finish.

2.4 Digital Signature

A digital signature is an electronic equivalent of the traditional signature. Typically, in the electronic transactions, the entities bind themselves to the transaction using the digital signature. The digital signature must fulfill certain requirements to be considered valid [6].

1. Authenticity. After the signing process signature must convince the verifier that the signature is authentic and the signer commits to the signature.
2. Unforgeability. Only the signer should be able to do an authentic signature.
3. Non-reusability. The signature must be unique in a sense that the same signature could not be used in any other transaction.
4. Unalterability. The signature must provide protection against alteration so that the signature cannot be invalidated after the signing process.
5. Non-repudation. The signer cannot afterwards deny that he or she has signed the subject.

Finnish law sets similar requirements for signatures to be valid in legal manner [7]. A material used to create a signature must be unique and stay confidential, e.g., private key used in the signing process must not be compromised. One should not be able to derive the signing material from any other information. The signature must be protected from forging and the signer must protect signature material from the use of any other entity. The digital signature can be used in a judicial act where a traditional signature is used. The signature should be based on the certificate recommendations of the law and fulfill previous conditions though it is considered legally valid even when all the conditions are not met. At the moment Finnish authorities do not offer an adjudicator service

for non-repudation conflict cases. This makes the legal value of collected evidence uncertain.

2.5 E-Commerce

Electronic commerce is performed more and more with digital products. In a traditional scenario, one goes to a shop where he or she can physically examine the item before buying. If one buys the item, the payment is typically made with money or credit card. When using the first one, shop physically gets the payment. It is possible to verify the payers' identity before accepting the payment. However, when using cash, this is quite rare procedure and more common with credit cards. In the credit card case, the credit card company typically provides assurance that the shop will get the payment and it is possible that the credit card company takes the responsibility in possible fraud cases.

In the electronic commerce situation, involved parties do not typically interact physically. In an example situation customer wants to buy a product, but does not want to pay before receiving it. On the other hand, the shop does not want to deliver the product before it gets the payment. In both cases, one entity must trust to the other to finish the transaction. If the customer misbehaves, he or she gets the product and the shop does not get the payment, or in the case the shop misbehaves, it gets the payment, but the customer does not receive the product.

The need for micropayment may occur when, e.g., a customer wants to use a WLAN hotspot. The customer pays to the provider beforehand. If an error occurs during the usage, the customer will lose some of the payment since he or she cannot use the service. So a mechanism similar to a phone tick system is needed to provide fair usage for this kind of scenario.

2.6 Contracts

When two parties communicate or make a contract, it is possible that afterwards one or both entities may deny being involved in the contract or the agreement. Similarly, one entity may deny its participation in the communication. In traditional cases entities are bound to the agreement with a signature. Typically, the signer's identity is also verified in the signing situation. Help of a notary may be used to satisfy the legal issues. Typically there can be also requirements for the authenticity, integrity and confidentiality in the communication. To satisfy the authenticity needs, non-repudiation procedures are required. For integrity and confidentiality needs some other procedures are needed. It should be noticed that non-repudiation mechanism solves only the problems related to the agreement phase. Negotiation and preparation of the terms of the contract may require further procedures and protocols.

3 RELATED TECHNOLOGIES

In this chapter, some protocols which can be used to achieve non-repudiation are introduced. The most important ones are Host Identity Protocol (HIP) [8] and authentication, authorization and accounting (AAA) protocols. Remote Authentication Dial In User Service (RADIUS) [9] protocol is examined more closely. Some other technologies were studied also. The other tools for non-repudiation are hash chains and Simple Public Key Infrastructure (SPKI) certificates.

3.1 Host Identity Protocol

Basically, HIP is a key exchange protocol [8]. HIP introduces some additional features to traditional key exchange protocols, e.g., Internet Key Exchange Protocol (IKE) [10]. However, the key exchange functionality is simplified from the IKE to provide a lightweight solution. The key elements are functionality to separate locator and identity information, end-to-end encryption and authentication, mobility, multihoming and interoperability between both IP families. The protocol contains mechanisms to provide protection against Denial of Service (DoS) and Man in the Middle (MitM) attacks. HIP establishes an IPsec protected connection between the end points [8].

3.1.1 Cryptographic Namespace

HIP performs the so called location and identity separation. For this separation HIP introduces a new namespace, called Host Identity. Host Identifier (HI) is an endpoint identifier for HIP connection. HI is basically a public-private key pair. Currently, the HIP implementations must support Rivers Shamir Adleman algorithm (RSA) [11] and should support Digital Signature Algorithm (DSA) [11]. Other algorithms may also be supported. Using host identities as the endpoint identifiers improves the security of the communication between entities. Host Identifiers are computationally expensive to forge which provides protection for the host's identity.

For actual use Host Identity is represented in hash format. Host Identity Tag (HIT) is a 128-bit long cryptographic hash which contains part of the original Host Identifier. HIT is a special type of Overlay Routable Cryptographic Hash Identifiers (ORCHID) [12]. Part of the HIT comes from the ORCHID section and rest from the actual Host Identity. ORCHID is intended to be used as purely endpoint identifier, without any locator functionality. Now HIT's can be used as regular IPv6 addresses with a low collision probability [8]. However, they are non-routable because of the ORCHID part. ORCHID part

is a IPv6 prefix, which is allocated non-routable in the IPv6 address space by Internet Assigned Numbers Authority (IANA).

3.1.2 Host Identity Layer

One of the Internet Protocol (IP) problems is that an IP address contains both the identity (endpoint, host) and the locator (routing label) information. HIP adds a new layer to the original TCP/IP stack. Position in the stack and the new endpoint identifiers are described in Figure 3.1. In the new layer model HI acts as an endpoint identifier. Host identities are non-routable, so IP addresses are still used as location identifiers for routing packets.

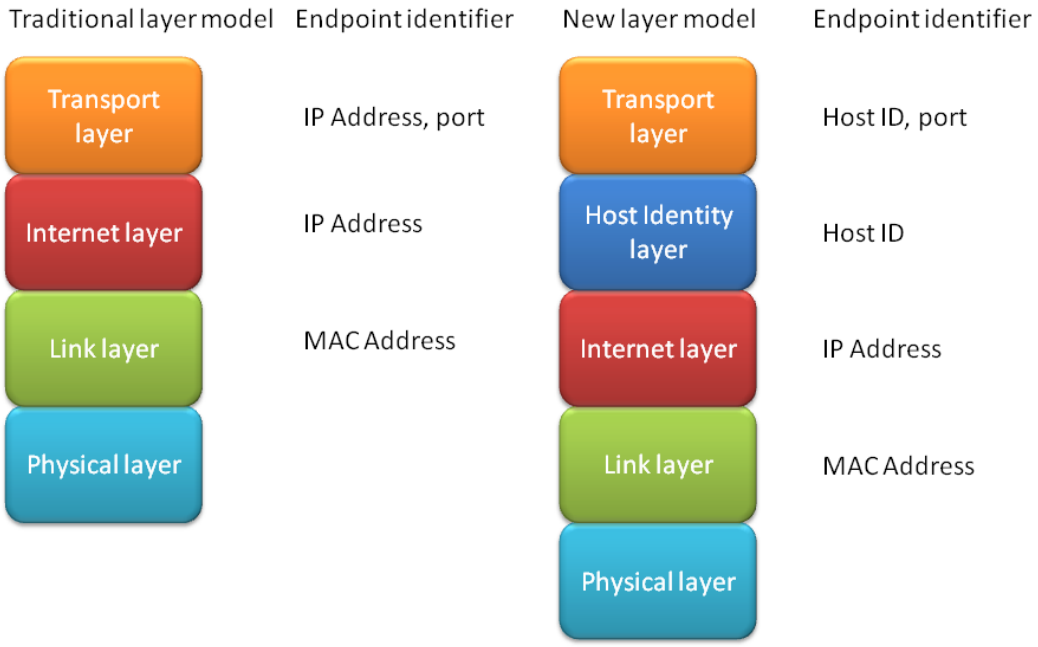

Figure 3.1. *Position of Host Identity layer in layer model*

3.1.3 IPv4

HIP supports both IPv4 and IPv6 protocols. For the IPv4 128-bit representation of HI cannot be used. A method called Local Scope Identifier (LSI) is used to provide an API support for legacy IPv4-only applications. LSI is a 32-bit presentation of the HI, which makes it shorter than HIT, but as a disadvantage it works only in a local scope because of a greater collision probability.

3.1.4 Mobility and Multihoming

The Locator/identity split gives a possibility to change the underlying IP address while the end-to-end connection stays online. When host's IP address changes, HIP has a built-in mechanism to update existing security associations with the new address. The

same mechanism is used in the multihoming situation, where a host has multiple interfaces for the Internet connectivity. It is possible to choose specific interface, which is used to establish and use the HIP connection.

3.1.5 Base Exchange

A base exchange between HIP capable hosts contains four phases which are presented in Figure 3.2. An entity, which begins the interaction, is called Initiator and other entity is Responder. First, Initiator signals willingness to use HIP by sending a trigger packet I1 to Responder. I1 packet contains Initiator's HIT and may contain Responder's HIT and additional parameters. There is a possibility to use a so called opportunistic mode, where Responder's HIT is not known.

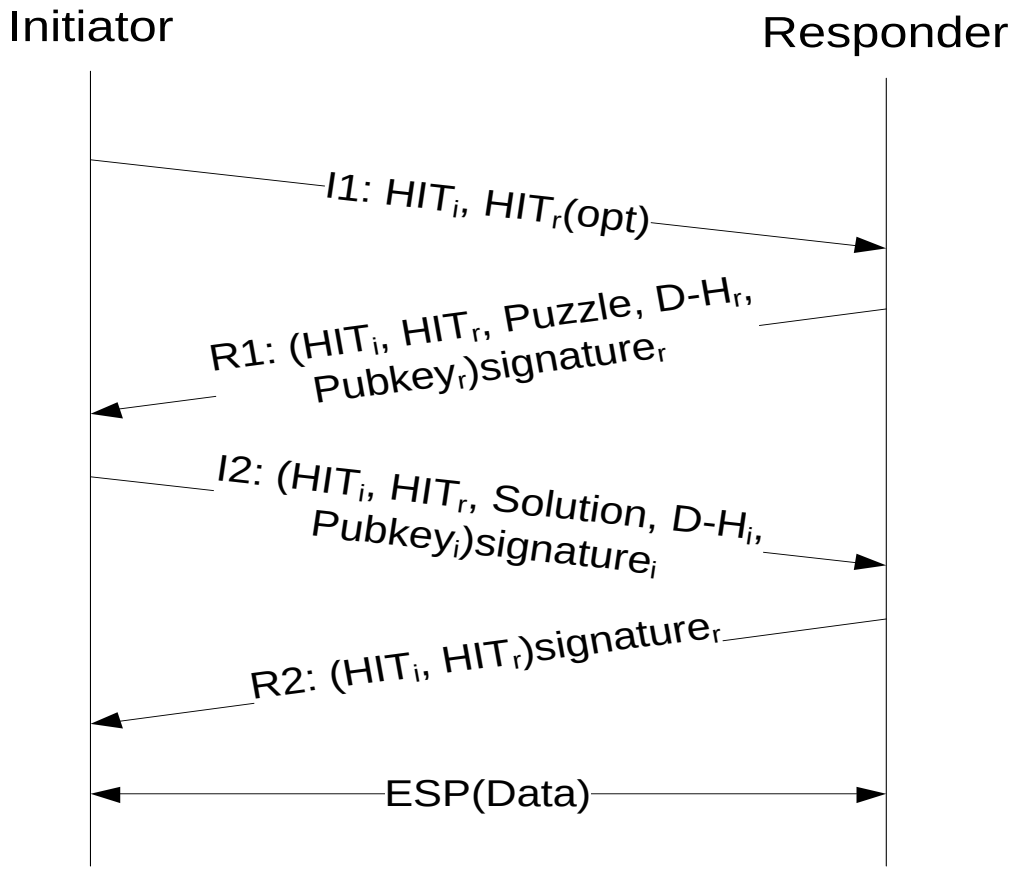

Figure 3.2. *HIP Base exchange.*

After receiving I1, Responder sends pre-created R1 packet to Initiator if it wants to continue base exchange. R1 contains HITs of both entities and a puzzle to Initiator. The puzzle is a cryptographic challenge and it is used for the denial of service protection. Initiator must solve the puzzle to continue base exchange. R1 contains also initialization parameters for Diffie-Hellman (D-H) key exchange [14] and information about cryptographic algorithms, which Responder supports. Responder's public key is included as

the final mandatory parameter. All of these parameters are protected with Responder's public key signature.

After solving the puzzle Initiator creates I2 packet. I2 contains earlier mentioned HITs, solution to the puzzle which was in R1, Diffie-Hellman information, cryptographic algorithms preferred by Initiator, and Initiator's public key. All these parameters are covered with hashed message authentication code (HMAC) and Initiator's signature. Key used with HMAC is obtained from the D-H key exchange.

Responder finalizes the base exchange by sending R2 packet which contains HIT's covered with HMAC and a signature. After that both entities consider the HIP connection established and data protected with Encapsulating Security Payload (ESP) [15] can be transferred between entities.

3.1.6 IPsec

IPsec architecture specifies protocols to establish a secure connection through an insecure network. The architecture contains four major parts: protocols for securing data, security policy and association handling, key management and exchange, and algorithms used for authentication and encryption. The original IPsec supports two modes: transport and tunnel. In the transport mode transferred data can be protected with an authentication or encryption. Additional fields are added to a regular IP header in the transport mode. In the tunnel mode a secure IP in IP tunnel is established between the endpoints. The original IP packet is encapsulated as the payload of the outer IP packet. [16]

HIP utilizes a new IPsec mode called Bound End-to-End Tunnel (BEET) [17]. BEET is a combination of IPsec transport and tunnel modes. Advantage is that BEET mode can pass Network Address Translations (NAT) while the transport mode cannot and there is less overhead than in the tunnel mode. Inner addresses are fixed addresses, in this case HITs, and outer IP addresses are used for the normal routing and therefore can be changed on the fly.

3.1.7 Host Identity Protocol for Linux

Host Identity Protocol for Linux (HIPL) is an open source Linux implementation of HIP which operates in Linux user space [18]. It is currently the most developed HIP implementation. The HIPL architecture consists of two main modules. HIP Daemon is the actual Linux daemon module which communicates with the IPsec stack. HIP Daemon handles all the base exchange related functionality and keeps track of open HIP connections.

The other important module is HIP Firewall, which is an iptables based firewall solution. It can be used to filter HIP connections. Modules communicate with each other using an internal communication mechanism, which is based on HIP sockets. HIP socket is a new socket type registered to Linux kernel. Use of the HIP Firewall is optional, if it is not in use, HIP Daemon communicates with kernel. HIPL contains a command line based control application called HIPconf. It uses the HIP internal communication mechanism to configure and set parameters for HIP Daemon.

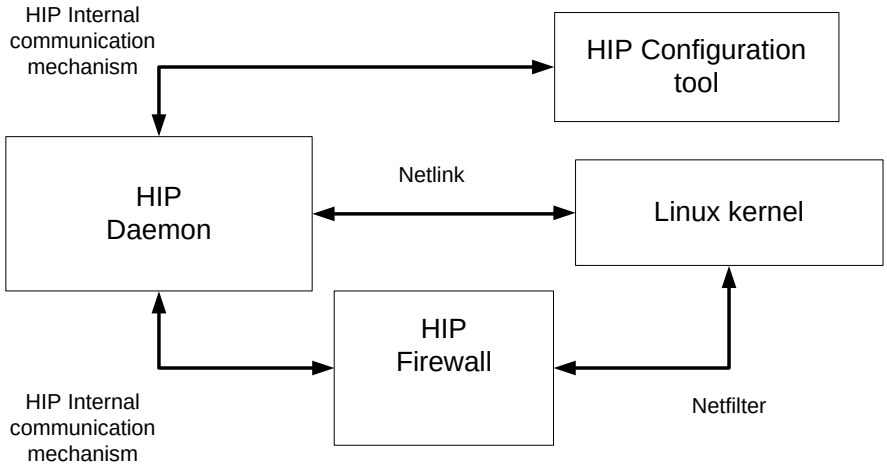

Figure 3.3. *HIPL Architecture.*

When an application wants to establish a HIP connection, it finds out opposite side HIT from the modified Domain Name System (DNS) server. After receiving HIT, the application requests a connection establishment from HIP Daemon, which initializes base exchange and negotiates a security association in the BEET mode. HIP Daemon uses the DNS server to resolve HIP-IP pair for the network level connection. The application uses HIT's as a source and a destination address for packets. The IPsec layer transforms HIT's to negotiated tunnel's Security Parameter Index (SPI). IP header is also replaced with the BEET addressing. Figure 3.4 describes HIPL interaction with Linux.

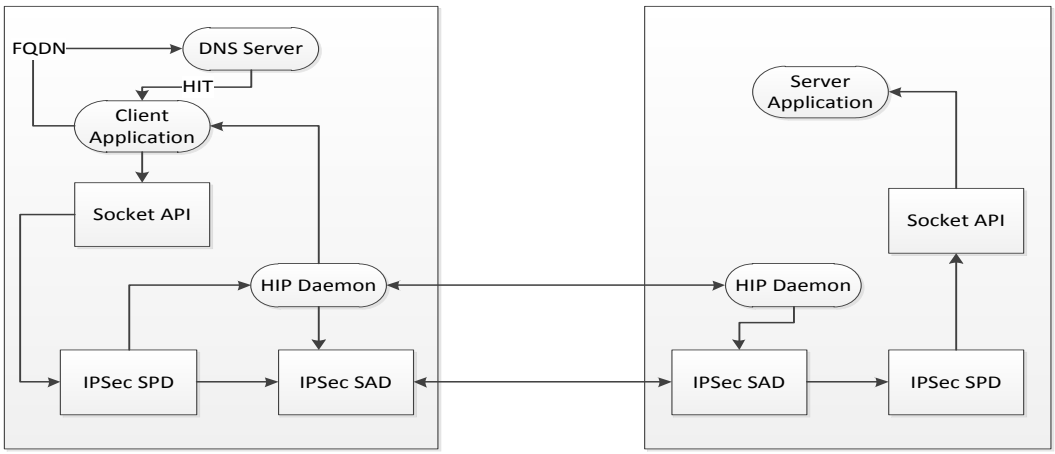

Figure 3.4. *HIPL interaction with Linux. [18]*

3.2 Authentication, Authorization and Accounting Protocols

Though HIP provides some security properties, a few enhancements are required to satisfy the needs of non-repudiable service. In a typical roaming scenario user is roaming in a foreign network and the foreign operator must charge the user's home operator after usage. This requires user authentication, authorization for the services, and accounting information about resource consumption.

Authentication is a procedure where entity's identity is verified. The entity must provide some information as proof of one's identity, e.g., a password or response to a challenge which is verifiable by the authentication server. This information is compared with an access database and authentication is confirmed if there is a match.

Authorization defines permissions for an earlier authenticated entity. It contains rules and restrictions what actions an entity can perform. Typically, authorization is done simultaneously with the authentication, but it can also be a separated process. It is possible that non-authenticated entity is authorized to perform some actions.

Accounting contains acts which are made to gather information about the resources consumed by the user. This information can be later used as evidence related to communication. Typically, accounting data can be billing information or used for auditing security issues. The accounting data can also be used to estimate future load and resource consumption of the system. [19]

The purpose of AAA protocols is to provide these above-mentioned services. Typically one protocol can handle all of these functionalities. Common AAA protocols are RADIUS [9], Terminal Access Controller Access-Control System (TACACS) [20], and Diameter [21]. In a basic scenario AAA client functionality is deployed to a Network Access Server (NAS), which acts as an access controller for the end-users. When end-users connect to the NAS device, it sends end-user information to the AAA server which informs the NAS about further actions. After or during network access the NAS device provides accounting information about the connection to the AAA server. The AAA server typically contains a user database which is used to perform AAA actions to end-users attached to the NAS.

3.2.1 RADIUS

RADIUS protocol was originally developed to provide AAA functionality for dialup connection providers. It has been further developed to fit the needs of, e.g., a WLAN usage. RADIUS is a client server based protocol where the client side is deployed to a NAS. One of the advantages is that RADIUS is a very flexible protocol which supports multiple authentication mechanisms.

Base of RADIUS is a connectionless and stateless protocol, which works on top of User Datagram Protocol (UDP). It has very wide support in NAS devices. RADIUS supports proxy functionality between servers, which makes it ideal to use between different organizations. This causes also a shortcoming, because all the data must be transferred through the proxy hierarchy to the very endpoint server. In this proxy scenario it is also possible that the first RADIUS server grants the authorization request while some other server might have more updated information about the subject being authorized. With basic RADIUS it is impossible to cancel already made decision. Because RADIUS is stateless all the necessary information must be in every packet sent. [9]

Because RADIUS operates over UDP, RADIUS protocol itself contains a retransmission mechanism. Many protocols rely on TCP in the retransmission functionality. On the other hand RADIUS AAA session must be completed fairly quickly, so even though TCP would be handling data transfer reliably it could take too much time.

One of the advantages of RADIUS protocol is the extremely flexible message format. It is based on Type-Length-Value (TLV) mechanism, where each parameter has type, value and length. RADIUS messages can contain improved TLV field called Vendor Specific Attribute (VSA), which offers possibility to encapsulate various kind of parameters inside RADIUS messages. RADIUS VSA Attribute is described in Figure 3.5. Vendor-id is identifier for the used VSA. Vendor type can be used for more specific information about the attribute. Vendor length indicates the length of the whole VSA parameter. Attribute-Specific section can contain multiple Attribute-Value pairs. Attribute data length is restricted to 255 bytes.

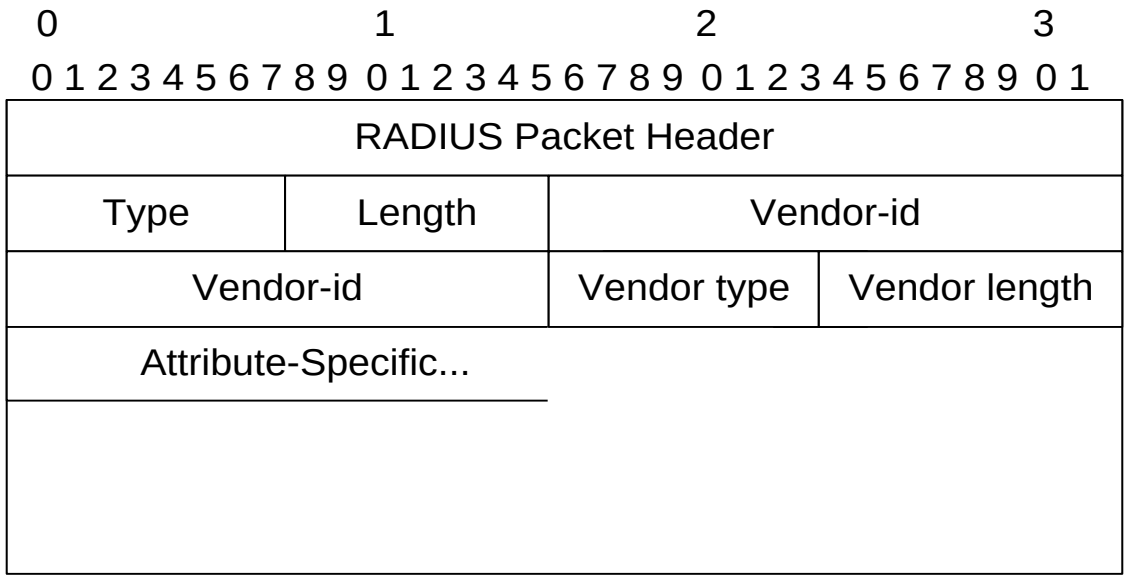

Figure 3.5. *RADIUS VSA-Attribute.*

Packets are authenticated with a pre-shared secret which every entity in the same RADIUS set must share. The pre-shared key is only used for message authentication. RADIUS does not encrypt packets; only the user password is MD5 hash of the concatenation of the pre-shared secret and password, when using Password Authentication Protocol (PAP). To maintain security in the communication, RADIUS can be deployed only to a network which is secure from end to end. Privacy is also weak, since packets are not encrypted and there is no protection for replay attack. RADsec [22] specification is being developed to provide better security when using RADIUS over an insecure network.

3.2.2 RADIUS Roaming

RADIUS Roaming can be used in a situation, where the client attaches to a foreign network's NAS. The NAS sends an access request to network's local access controller, which sends an RADIUS authentication to client's home organization RADIUS server. Request can be relayed through various RADIUS servers.

Finnish University and Research Network (FUNET) offers a RADIUS roaming service to Finnish Universities and organizations. In FUNET RADIUS roaming a user, which is a member of FUNET RADIUS roaming organization, attaches to a public access network which is managed by other FUNET RADIUS roaming organization. An access request is delivered to the FUNET Root Roaming Server, which delivers the request to roaming user's home organization RADIUS server. An answer to the request is relayed back to the access controller, which decides if roaming user is allowed to connect to the network. [23]

3.2.3 Diameter

Diameter protocol aims at responding to the AAA challenges of new Internet technologies and capability requirements. Diameter is used in the IP Multimedia Subsystem (IMS) architecture and some other 3GPP applications to provide a AAA functionality between entities [24]. It also tries to improve some shortcomings of the RADIUS protocol.

Diameter is a connection and session based protocol which supports TCP and SCTP transmission protocols. Diameter supports capability negotiation so the client and the server can try to find a service level which satisfies both. RADIUS requires static configurations of peers while Diameter allows dynamic peer discovery [25]. The most important advantage in Diameter is the attribute size. It supports 16,777,215 byte length of parameters while RADIUS supports only 255 bytes. This gives a good flexibility for the vendor specific attributes. Diameter offers also a better reliability while RADIUS suffers problems in congested networks [26].

3.2.4 TACACS+

TACACS+ is a AAA protocol developed by Cisco [27]. It aims to fix some shortcomings of the RADIUS protocol and add more security. TACACS+ employs TCP as a transmission protocol and is session based. The most important security improvement is that TACACS+ encrypts the whole packet body while RADIUS encrypts only the passwords. A shortcoming is that the encryption is made with a secret key and MD5, even though MD5 has some security vulnerabilities [28]. The whole packet body encryption makes TACACS+ more feasible than RADIUS in insecure networks, if attention is paid to the MD5 vulnerability. Another improvement is separation of AAA functionality while RADIUS combines the authentication and authorization. This allows more advanced authorization scenarios where non-authenticated entities may still have some privileges. The biggest shortcomings of TACACS+ are availability and distribution. It is not as widespread as RADIUS and available mainly in Cisco devices only, and every entity attached to the TACACS+ infrastructure must share the same secret. TACACS+ is lacking integrity checks in the accounting packets which make them vulnerable to tampering.

3.3 Hash Chains

Use of hash chains was introduced in [29]. Cryptographic hash function is a one-way function which is easy to apply to a seed value(s), but computationally very expensive to reverse. A hash chain is created when the cryptographic hash function is applied to the results of previous hash functions. Equation (1) shows the creation process of a hash chain.

$$s, H(s), H^2(s), \dots H^{N-1}(s), \dots \tag{1}$$

Last piece of the chain is called an anchor value. Now it is easy for the receiver to verify, if the value belongs to the hash chain, by applying the hash function to the previous value. On the other hand, it is computationally difficult to perform the verification procedure without knowing the seed value. Equation (2) shows the verification process.

$$H_{N-1} = H(H_N) \tag{2}$$

Hash chains can be used in micropayment solutions [30] as well as in the onetime password (OTP) schemes [32] and many other purposes including authentication [31].

3.3.1 Infinite Length Hash Chains

One of the problems using traditional hash chains in micropayment solutions is the finite length. Typically the duration of the service usage is not known beforehand which makes it difficult to estimate the right length of a hash chain.[33] proposes solution

where a public-key technique is used to create an infinite length hash chain. This means that the chain length can be increased without restarting. Equation (3) defines how the infinite length chain is constructed by applying a public-key based algorithm (*A*) and a private key (*d*) to a secret seed value (*s*).

$$s, A(s,d), A^2(s,d), \dots A^{N-1}(s,d), \dots \tag{3}$$

The verification process goes similarly, the public key algorithm used in chain creation, is applied to a chain piece. Equation (4) defines the verification process where (*P*) denotes the received piece and (*e*) chain creator's public key.

$$P_{i-1} = A(P_i, e) \tag{4}$$

It should be noticed that the chain creator must not send the first created chain piece or the secret seed value is revealed to the verifier. This scheme requires the chain creator's public key to be transferred to the chain verifier before the chain use can begin.

3.3.2 Hash Chain Tree

An unbalanced one-way binary based hash chain tree is a special application for hash chains [34]. In the hash chain tree one hash chain acts as a secret seed value for the other chains. This helps to save space when a large amount of pieces is need. Since every piece of the hash tree can be generated from a single root value, the used device does not have to store every value. Figure 3.6 shows an example of the unbalanced one-way binary based hash tree. The first chain in the horizontal direction is so called an anchor chain, which contains anchor values for each sub chain. The actual used chains are generated from these anchor values.

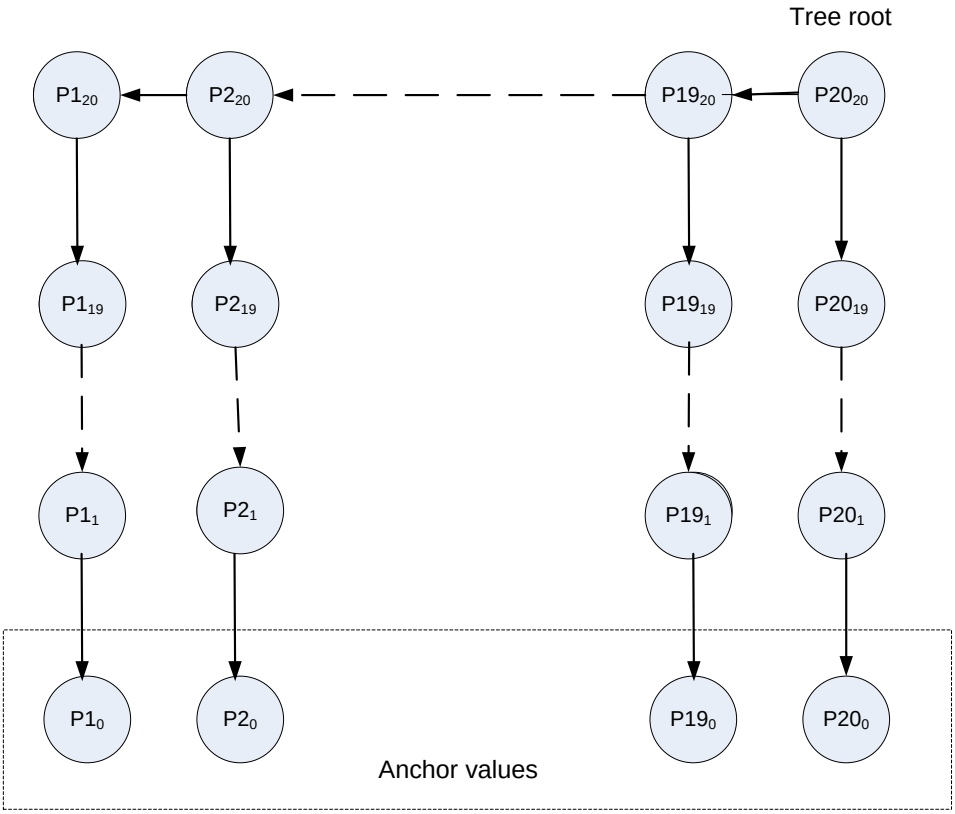

Figure 3.6. *Unbalanced one-way binary based hash chain tree.*

In a micropayment scenario it is easy to send multiple anchor values under a single signature for the service providers. While this creates a little overhead it saves computation power since cryptographic verification is not needed when switching to the next chain.

3.4 Public Key Certificates

Public key certificates are used to bind entity's identity or some attributes to entity's public key [35]. A basic public key certificate contains only entity's identifier, e.g., name or other verifiable subject, entity's public key, issuer's identifier and signature, which performs the actual binding. Certificates can be self-signed or TTP, typically called Certificate Authority (CA), provides a signing service for certificates. TTP may provide a verification service for the identities in the certificates which TTP has signed.

Self-signed certificates are based on a web of trust. Certificate holders provide trust for each other since typically there is some strong binding between the holders. For example if Alice has a certificate which is signed by Bob. Alice does not know Bob, but Alice knows Carol, who knows that Bob can be trusted. Now Alice can also trust on Bob, because she knows Carol, who convinces Alice that Bob is a trustworthy person. Ultimately, security is based only on trust. Compared with CA signed certificates, identity provided in CA certificates hold more trustworthy status than self-signed certificates, depending on what kind of verification CA performs to the certificate signing requester before signing the certificate.

Typically, certificates provide some additional information about the entity. This information can be used in advanced authentication and authorization situations. Commonly certificates have a validity period after which they have to be renewed. TTPs may offer also a certificate revocation service. Revocation is used when the key pair used in the certificate is compromised, or an entity which is certified performs some actions after which it cannot be trusted. The revocation functionality requires typically online TTP.

3.4.1 X.509v3 Public-Key Infrastructure

X.509 Public-Key Infrastructure was developed by ITU-T. It is the most widely spread and used PKI system in the electronic transactions. X.509 specifies certificate structures, the revocation process, and certification path validation procedures [35]. X.509v3 is the latest version of X.509 family specification and brings some new features and improvements to the older ones. X.509 system uses a global namespace while, e.g., SPKI uses only application domain specific namespace. The global namespace and certificate complexity causes problems in scalability. Figure 3.7 shows an example of X.509v3 certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 103 (0x67)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=FI, O=Tampere University of Technology, OU=IT Management, CN=TUT
Internal Root CA
        Validity
            Not Before: Aug 20 06:39:13 2001 GMT
            Not After: Aug 18 06:39:13 2011 GMT
        Subject: C=FI, O=Tampere University of Technology, OU=IT Management, CN=TUT
Internal Root CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b1:76:a0:67:24:86:06:e1:e5:a6:7d:34:62:a9:
                    81:18:32:7a:81:9f:c9:aa:30:55:8a:43:df:1e:a7:
                    11:a6:e8:7c:3d:d0:f3:ce:bc:87:71:7a:11:a6:39:
                    81:ad:23:b3:33:f7:4a:c6:f5:fc:a1:5c:95:ca:76:
                    c6:dd:88:4b:3d:07:04:63:c3:84:7a:07:ee:2c:7c:
                    e3:90:bb:90:60:78:09:0d:cd:5b:a1:9e:bb:3a:f4:
                    eb:99:a1:50:a7:5e:7b:d1:a1:aa:b2:58:bf:e8:01:
                    6d:82:60:f2:0d:7c:ac:99:80:c4:c6:73:6f:da:dc:
                    ad:b2:74:b0:69:b4:01:99:a9
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        2d:ef:65:71:ec:82:2d:91:df:43:49:3c:eb:5a:75:e1:eb:9d:
        cf:61:de:db:28:4d:3f:3b:d8:42:10:45:68:31:85:9c:04:27:
        cd:c8:a1:7c:7c:fd:d8:5c:62:0b:81:e4:3d:3b:6f:9f:65:63:
        a2:7b:84:85:76:5f:31:07:87:9a:d5:20:cd:b4:b5:5e:2b:4f:
        ab:42:5b:3c:24:97:06:21:50:9a:94:ee:d4:73:87:b6:3d:85:
        10:92:bc:81:13:e0:33:92:e2:a5:31:30:1d:b8:56:79:79:7c:
        35:b2:11:20:75:fd:f4:19:87:b4:92:19:0a:60:80:9e:aa:26:
        96:18
```

Figure 3.7. *Example X. 509v3 certificate formatted for readability.*

In an example situation user wants to use X.509 certificate as a proof of his or her identity. As mentioned before, a public key certificate binds user's identifier to a public key. X.509 structure allows only binding user's actual name, which is used as a identifier, to the public key instead of identity. When users with the same names exist, it is difficult to distinguish the users from each other.

3.4.2 SPKI

Simple Public Key Infrastructure (SPKI) defines a simple certificate structure for decentralized and distributed authorization scenarios [36]. SPKI defines two basic types of certificates, naming and authorization. Figure 3.8 shows an example of SPKI certificate.

```
(
(cert
(issuer(hash sha1 (abcd)))
(subject(hash sha1 (efgh)))
(validity
(not-before 2011-03-11_12:00:00)
(not-after 2011-03-21_12:00:00)))
(signature
(rsa-sha1(abcd))))
)
```

Figure 3.8. *Example SPKI certificate.*

Naming allows binding names to public keys. Naming is valid only in the specific domain where SPKI is used. Authorization certificates include the possibility to delegate some or all of the original authorizations. This makes it possible to create chains of authorization where original authorization propagates through the chain. SPKI certificates transferred from machine to another must be encoded in canonical S-expressions [36]. Canonical S-expression is a form of S-expression, where length of the element is coded before the atom. Figure 3.9 shows example canonical S-expression.

```
(7:example10:expression)
```

Figure 3.9. *Example canonical S-expression.*

Length is in the ASCII decimal format and a colon is used to separate the length from the actual element.

3.4.3 KeyNote

KeyNote introduces a trust management based solution to perform authorization. It uses credentials to decide whether the user is allowed to perform some actions. In the example case when using KeyNote, used applications must provide a KeyNote Application Programming Interface (API). After an application has received some action request from the user, the application sends a KeyNote query to local the KeyNote Interpreter. This interpreter interprets the action request and asks authorization from the credential

management entity. Based on the users' credentials, the action is granted or denied. Two main advantages of KeyNote are easy importability and flexibility in the message syntax. The syntax is less structured than, e.g., SPKI or X.509 so it allows more expressive notations.
Figure 3.10 shows an example of KeyNote assertion statement. [37]

```
KeyNote-Version: 2
Authorizer: "rsa-hex:1234abcd"
Licensees: "dsa-hex:5678efgh"
Comment: "Authorizer offers WLAN service with time or volume based billing"
Conditions: app_domain == "wlan" -> "true" && currency == "EUR" && amount ==
"0.15" && date < "20110315" -> "true";
Signature: "sig-rsa-sha1-hex:4321cdea"
```

Figure 3.10. *Example KeyNote assertion.*

KeyNote can be used in various scenarios to provide access control and authorization. [38] shows example solution where KeyNote is used with IPsec to provide access control. Due to the flexibility of KeyNote syntax, it is also suitable to be used in micropayment solutions [39].

4 SYSTEM REQUIREMENTS

When the implementation phase began, some functional and non-functional requirements were set for the implementation. The functional requirements were related to providing the non-repudiation properties while the non-functional requirements focused on the user experience and security issues. The non-functional requirements were also used to evaluate the implementation.

4.1 Functional Requirements

Non-repudiable service usage implementation (NoRSU) focuses on three main goals. The first goal is to make it possible to offer different kind of non-repudiation services using HIP. These services can be for example a WLAN usage or streaming service. The second goal is to bind the service provider and user undeniably to the service usage. The third main goal is the gathering of evidences about the communication between entities. It should be also possible to choose whether to operate using NorSU implementation or traditional HIP. NoRSU incapable Responder should be capable of performing the traditional base exchange with NoRSU capable Initiator. Operation must be possible vice versa where Initiator is incapable to use NoRSU to provide a backward compatibility with vanilla HIP implementations.

4.2 Non-Functional Requirements

The most important non-functional requirement is to provide non-repudiation for the communication. All transactions and agreements made between entities must be bound to their identities. Also the evidence of all transactions must be gathered.

4.2.1 Non-Repudation and Fairness

In the base exchange Responder must commit to a service offer with a signature. If Initiator decides to agree the offer, it must commit to the agreement with a signature. Either one of the entities must not be able to deny involvement in the agreement afterwards. After the base exchange both entities must be bound to the sent messages until the session tear down. Either one of the entities must not be in a considerably weaker position than the other at any point in the transaction. If either one of the entities feels its position unfair, it must be able to withdraw from the session, without noticeable losses.

4.2.2 Expandability

The implementation must establish a modular and easily expandable framework for the non-repudiable service provisioning. Because of the general nature of the framework, the main components must be easily changeable and modifiable. The implementation should provide an interface to easily add and modify new certificate types and encodings. The implementation configuration files and statements should be flexible to support possible future parameters. Also the use of longer key lengths and new types for cryptographic algorithms and hash functions should be possible to implement easily. Charging schemes should be easy to customize to fulfill the needs of different service providing scenarios. The framework should also provide an interface for using and switching between different kinds of AAA protocols. There should also be an interface for local evidence handling. The possibility for saving evidence into files should be provided, which could be extended with a database connection in the future.

4.2.3 Compatibility

The framework should be compatible with existing HIP solutions, so no changes to the basic HIP functionality should be made. The compatibility includes HIP aware firewalls and middleboxes. Use of the NoRSU modes should require support only in the connection initiating and responding devices. The NoRSU functionality should be as transparent as possible to upper level applications, so no support for NoRSU is needed from the upper layers. However, there should be an API which is used to inform the user about the NoRSU related information and gather user's approval or rejection to the service offers. No modifications should be made to the HIP API and legacy API.

4.2.4 Usability

The configuration and usage of NoRSU implementation should be simple. Configuration files must be in a simple attribute value pair format. The main functionalities of the implementation must be configurable; this includes hash chain lengths, certificate validities and used identities. The controlling of the implementation is done using an existing HIP configuration command line user interface. There should be information available for the user when NoRSU mode is enabled or disabled and possible error situations. This should be done using existing HIP debug information mechanism. The user should be able to adjust the amount of information given by the implementation, but the most critical error situations, which have effect on the service usage, must override existing settings. When the base exchange is finished successfully in the NoRSU mode, the user is not able to turn off the NoRSU functionality since this could compromise the fairness and non-repudiation properties.

4.2.5 Efficiency

Compared to the original HIPL, NoRSU modifications must not cause major lack of the efficiency including hardware and network requirements. This includes minimizing the bandwidth usage of the NoRSU related communication. Since the original HIPL does not have clearly defined hardware and network requirements, efficiency can only be measured in how many sessions one server machine can handle. This is difficult to verify without a large test environment, so the base exchange duration and control packet handling times must be used to estimate the efficiency. [40] shows some evaluation about the performance of several HIP implementations.

4.2.6 Security

The implementation must not weaken existing HIP security properties. This includes DoS protection and a secure communication channel between the participating entities. The solution must also fulfill security requirements which are derived from the non-repudiation functionality. This includes protection against denying being involved in the communication afterwards, protection against faking the identity of a participant, and fair execution of the base exchange. If the base exchange execution is compromised, implementation must try to ensure a fair position to all involving entities. In the actual payment phase, it must be computationally difficult to forge the units used to pay for the service usage.

5 DESIGN

The basic principle of the design was modularity. Existing HIP properties were used as much as possible to provide non-repudiation. New modules can be disabled so HIP can operate in the original mode. This chapter describes the main components of the implementation and interoperation between different components.

5.1 Architecture

The system architecture is presented in Figure 5.1. The component interaction is described in more detail in Section 0. Key components are HIP Daemon, RADIUS Daemon (RDaemon) and RADIUS Client (RClient). HIP Daemon is responsible for the HIP functionality. RDaemon handles the capsulation of RADIUS packets inside HIP packets. RClient is used to communicate with the actual RADIUS server. In the first phase, marked with 1 in Figure 5.1, the client initiates the connection to the server. If the normal mode is used, the base exchange continues traditionally. If the client wants to use the non-repudiable service, NoRSU mode is enabled in the phase one. If the TTP is not available, the base exchange continues as in the normal mode with NoRSU additions. If a HIP capable TTP is available and the server wants further verification of the client, it can establish a connection to the TTP.

In the phase two the server requests TTP's HIT from RDaemon. In the phase three the server begins the base exchange with the TTP. If the TTP is not HIP capable, the server goes to the phase four and connects to the TTP using RClient and the RADIUS protocol. If the TTP is HIP capable, in the phase four the server requests RADIUS communication from RClient. RClient creates a RADIUS message with requested parameters and sends it. When using RADIUS encapsulation RDaemon captures the message coming from RClient and transforms it to a HIP parameter in the phase five.

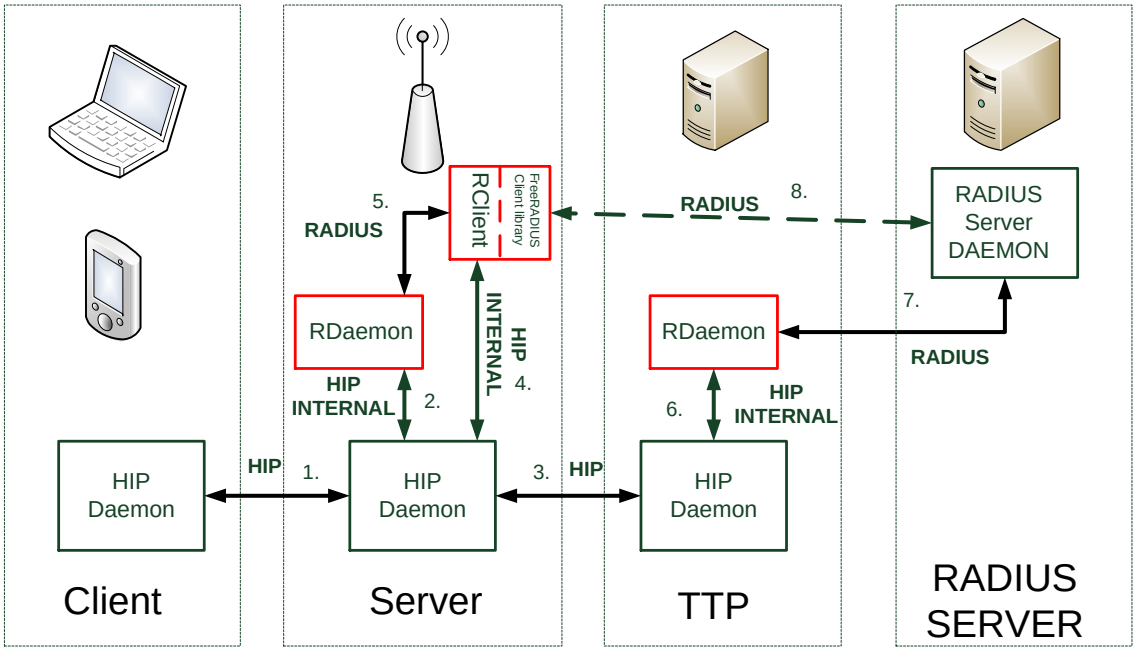

Figure 5.1. *NoRSU system architecture.*[41]

In the phase six TTP's HIP Daemon sends the HIP parameter which contains a RADIUS message for extraction. RDaemon extracts the message from the parameter and delivers message to RADIUS Server in the phase seven. The eight phase describes the logical connection between Server and TTP RADIUS services. HIP infrastructure offers delivery service for RADIUS messages.

5.2 Basic Modules

Additions to the core HIPL modules are presented in Figure 5.2. Four building blocks for the non-repudiation are added to HIP Daemon. The first block is the certificate handling module which creates, verifies, signs, encodes and decodes certificates used in the communication. It is extended from the basic HIP certificate module by adding support for S-encoding and efficient encoding, which is described in more detail in Section 6.3.

HIP Firewall contains a usage control module which is used in the volume based charging. The module keeps track of transmitted data of every NoRSU enabled HIP-connection and signals HIP Daemon when the set threshold is exceeded. The NoRSU control module is responsible for handling NoRSU configuration options.

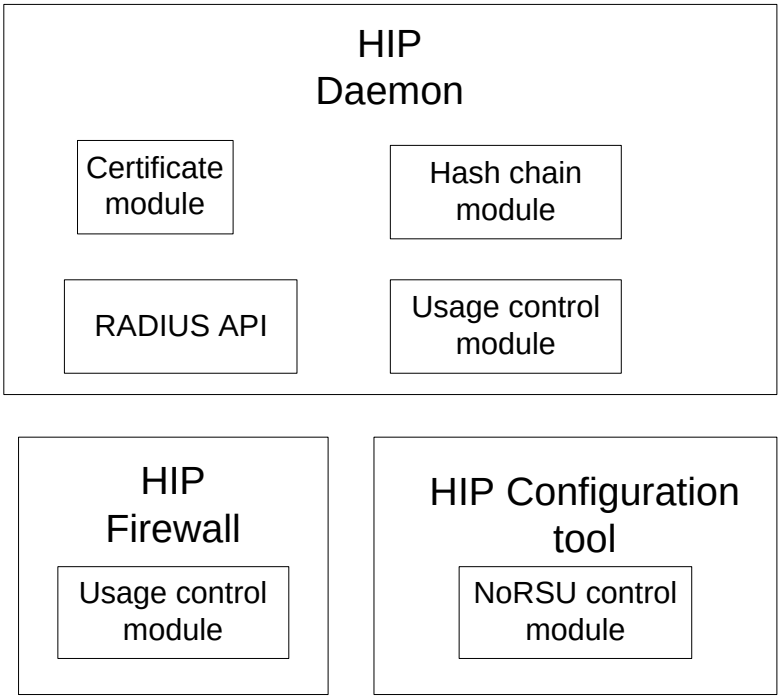

Figure 5.2. *Basic structure of the design.*

The hash chain module is responsible for creating and verifying hash chains. It also creates parameters for the hash chain renewal. Hash chain renewal is a procedure, where exhausted chain is replaced with new one, without interruption in the service usage. Hash chain module supports traditional hash chains and public key signature based infinite length hash chains. The usage control module contains functionality to monitor the non-repudiation process. It monitors the timers related to charging and creates mapping between the gathered evidence and security associations.

5.3 RADIUS Processing

RADIUS processing modules are presented in Figure 5.3. RDaemon contains a functionality to convert RADIUS parameters to HIP parameters and vice versa. RClient connects to RDaemon which encapsulates the received RADIUS message to a HIP Parameter. When RDaemon receives HIP Parameter which contains an answer to the message, it extracts the RADIUS message and returns it to RADIUS client. RDaemon also contains a TTP selector, which is used to select an appropriate TTP. TTP is used to verify the client side authorization certificate, which the client may supply in the base exchange. Different clients may have authorizations from different TTPs, so mechanism to choose right one is needed. RDaemon is only needed when using HIP to secure RADIUS messaging.

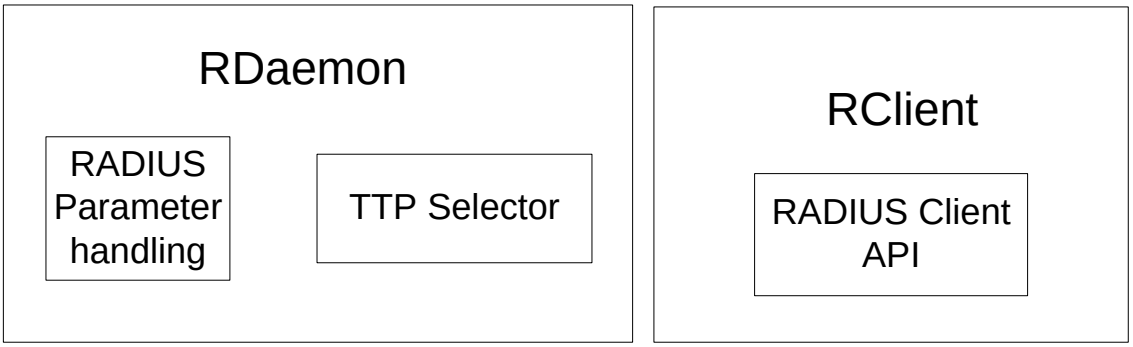

Figure 5.3. *RADIUS modules.*

HIP Daemon uses RClient to send RADIUS messages. RClient contains an API to communicate with RADIUS Client library. It also transforms NoRSU service attributes to RADIUS vendor specific attributes and RADIUS messages to NoRSU actions. For example, if a RADIUS message is AccessReject, RClient signals HIP Daemon to terminate the base exhange.

5.4 Operation Modes

The implementation can operate in several modes. This includes the NoRSU mode and the normal mode. In the NoRSU mode the system can operate in two different setups. The first setup contains an initiator and a responder. This is the most basic situation where the responder offers non-repudiable service and the initiator wants to use it. In the second mode the responder uses a TTP to get additional verification of the initiator. Communication between the responder and the TTP is done using a RADIUS protocol. The responder and the TTP can also establish a HIP connection where RADIUS is encapsulated inside HIP control messages.

5.4.1 Initiator Functionality in NoRSU Mode

On the initiator side, once the NoRSU mode is enabled an additional parameter to indicate the NoRSU is added to the I1 packet. The HIP Daemon configuration mechanism must support enabling and disabling the NoRSU mode. The user must be able to specify the contents of the NoRSU parameter via a configuration file. When receiving an R1 packet, the implementation must check if an offer parameter is included, determine the encoding used in the certificate and verify validity time and the signature. The initiator must support at least SPKI certificates in canonical S-encoding and efficient encoding [42] formats. If either one is invalid or missing, or the whole certificate is missing, the base exchange continues in a normal mode. The initiator must determine which charging type is used. At least traffic and volume based charging must be supported on the initiator side. In the traffic based charging support to monitor the amount of incoming and outgoing traffic of a specific HIP connection is added. HIP Firewall must also be capable of receiving threshold values, which defines how much traffic can pass through

with one token from the HIP Daemon and notify the HIP Daemon when the threshold value is exceeded.

When creating the I2 packet, the initiator must use the specified configuration file to create a response certificate. The configuration file contains a validity period of the response, the used hash chain and length, and HIT used in the certificate. The response certificate must use the same encoding that the offer certificate used. The initiator may also include a TTP authorization and delegation certificates in the I2 packet. If they are included, efficient encoding must be used. TTP and delegation certificates are specified in the separate configuration file. After accepting the offer and signaling the acceptance by sending the response certificate, the initiator must begin to monitor the hash chain piece sending interval condition. Pieces must be sent in the UPDATE packets and they must be encrypted.

5.4.2 Responder Functionality in NoRSU Mode

The responder supports three operating modes: a normal mode, a NoRSU mode and a NoRSU mode with AAA messaging. In the NoRSU mode when the responder receives an I1 packet with the NoRSU parameter it must include a service offer in the R1 packet. The responder may also include a TTP service authorization. If the I1 packet contains information about the desired service, the responder may follow the wish. Types of offered services, charging amounts and intervals, used encodings and service offer durations are specified in the configuration file. The responder must support at least SPKI certificates in canonical S-encoding and efficient encoding formats. If a TTP authorization is included, the efficient encoding is mandatory. Use of the S-encoding would cause the control packet to fragment. If the I1 packet does not contain the NoRSU parameter, the base exchange continues in the normal mode which is described in Section 5.4.4. If the I2 packet contains a response certificate, the responder must verify the hash of the offer, a validity period of the certificate and the signature. The response certificate, which the initiator creates, must also be in the same encoding format as the offer certificate. If any of these conditions are not met, the responder must continue the base exchange in the normal mode. If the response certificate is accepted, monitoring the charging must be initialized and in the volume based charging HIP Firewall must be configured similarly as on the initiator side. The configuration file must specify threshold values for how many and how long chain pieces can be missing. Figure 5.4 describes the base exchange in NoRSU mode.

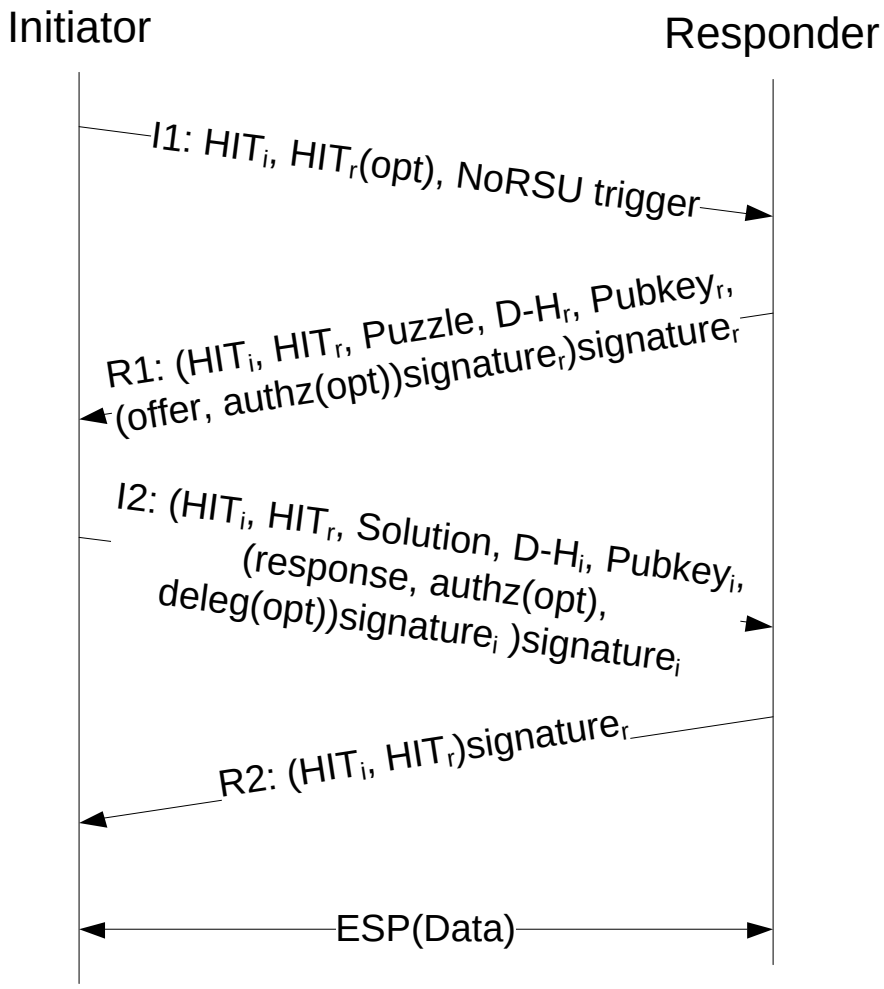

Figure 5.4. *Base exchange in NoRSU mode.*

5.4.3 Responder Functionality in NoRSU Mode with AAA Messaging

In the NoRSU mode with AAA messaging, after receiving the I2 packet, the responder contacts the AAA server specified in the configuration file. The server may ask further verification for initiator's identity and establishes an accounting session. If the AAA server returns a negative answer to the query, the responder must tear down the base exchange. The responder must support at least the RADIUS protocol for the AAA functionality. After connection teardown between the initiator and the responder, the responder must send used certificates and accounting information using AAA functionality to the AAA server. In the NoRSU mode the responder must save this information locally. Communication using the AAA server is described in Figure 5.5

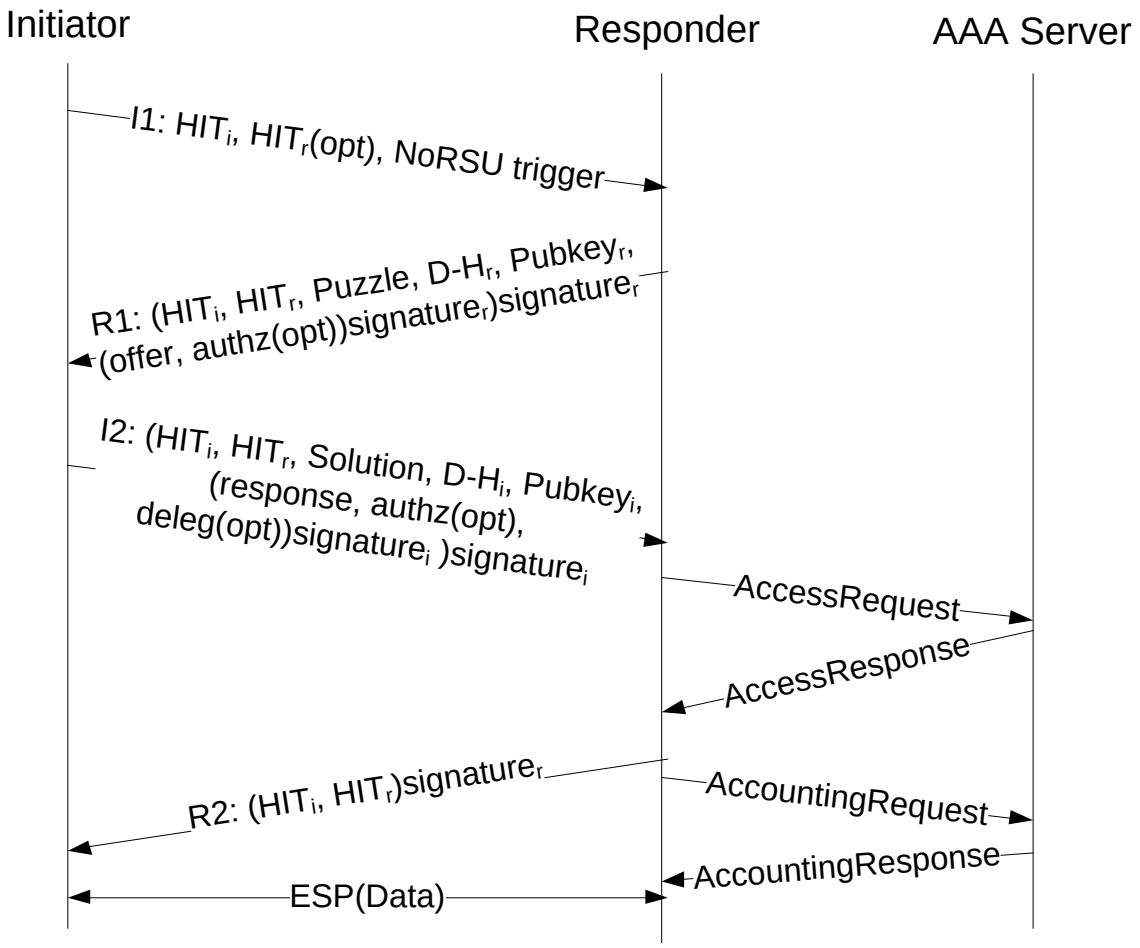

Figure 5.5. *Base exchange with AAA support.*

If the used network is insecure and the AAA provider is HIP capable, it is possible to establish a HIP session between the responder and the AAA provider. Communication goes like in the previous case, though the HIP responder must now relay AAA messages carried by HIP to the available AAA server. After the connection establishment, it may remain available to further AAA communication. The responder side HIP must be able to relay AAA requests over the second HIP connection. This functionality is described in Figure 5.6.

The responder may try to establish connection to several AAA providers during the base exchange. If any of them do not respond, it may fall back to a simple scenario where the AAA information is stored locally. The initiator may also include information about the AAA provider during the base exchange.

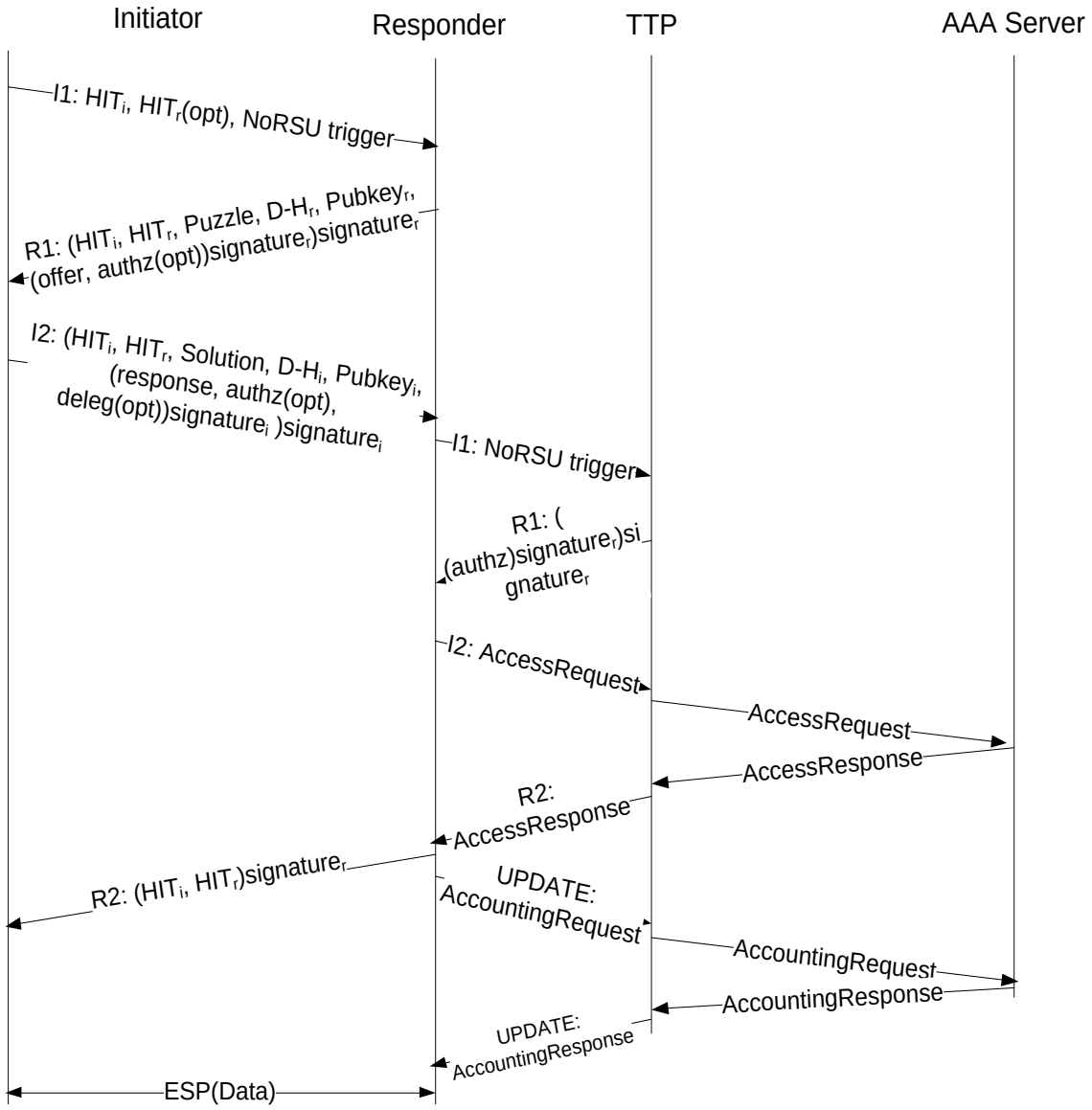

Figure 5.6. *Base exchange with HIP capable AAA.*

5.4.4 Normal Mode

In the normal mode the implementation must act as specified in [8]. A NoRSU unaware initiator must be able to establish connection to a NoRSU aware responder. It is not allowed to move from the normal mode to the NoRSU mode without establishing the base exchange first. Transition to the normal mode from the NoRSU mode during the base exchange must be supported in both the initiator and the responder side. The normal mode is only for maintaining compatibility. Transition to the normal mode will immediately end the service usage.

6 IMPLEMENTATION

In this chapter, most of the key implementation details are described. The implementation was done using C programming language and Ubuntu Linux 9.04 as development environment. The implementation is based on the HIPL 1.04 and it can be installed as an update to normal HIPL 1.04.

6.1 Host Association Database

Host Association Database (HADB) contains all data related to a one host association. Host association is a connection between two HIP hosts and which endpoints have common keying material. This was a natural place to store the NoRSU specific information. HADB is a single struct containing pointers to connection options. The following parameters were added to the struct. Table 6.1 contains certificate related variables.

Table 6.1. *Certificate related variables in HADB.*

Variable	**Type**	**Description**
hip_cert_spki_offer	struct	Storage for SPKI encoded offer certificate
hip_cert_spki_response	struct	Storage for SPKI encoded response certificate
hip_cert_eff_offer	struct	Storage for efficient encoded offer certificate
hip_cert_eff_response	struct	Storage for efficient encoded response certificate
hip_cert_eff_ttp	struct	Storage for efficient encoded TTP certificate
hip_cert_eff_dele	struct	Storage for efficient encoded delegation certificate

For the infinite hash chain support, separate variables were needed. Since only one type of hash chain can be used simultaneously, the same counter variables are used for both chain types. Table 6.2 lists hash chain related variables.

Table 6.2. *Hash chain related variables in HADB.*

Variable	**Type**	**Description**
norsu_hash_chain_values	unsigned char * array	Array to store pointers to unused hash chain value in the client side
norsu_hash_chain_values_switch	unsigned char * array	Array to store pointers to received hash chain switch parameters
norsu_rcvd_chain_values	unsigned char * array	Array to store received hash chain pieces the server side
sent_hash_counter	unsigned int	Counter for amount of sent hash chain pieces in the current chain client side
rcvd_hash_counter	unsigned int	Counter for amount of received hash chain pieces in the current in the server side
norsu_first	unsigned char *	Even hash chain piece in infinite length chain. Sent piece in the client side and received in the server side
norsu_second	unsigned char *	Uneven hash chain piece in infinite length chain. Sent piece in the client side and received in the server side

The general service related parameters are also stored in the HADB. These include type of used encodings, hash chain types, hash chain lengths, RADIUS specific variables and timers for service usage. Table 6.3 contains the service related variables.

Table 6.3. *Service related variables in HADB.*

Variable	Type	Description
cert_counter	int long	Counter used to monitor elapsed time, when using time based charging
cert_clock	int	Flag to store information, if time or traffic based charging is used.
norsu_encoding	int	Flag to store information, if SPKI or efficient encoding is used.
norsu_chain_len	int	Used hash chain length
norsu_duration	int	Duration of the service usage
rad_state	int	Flag to indicate RADIUS message exchange state
ttp_ha	int	HIT of the used TTP
hash_total	unsigned int	Total amount of received and sent hash chain pieces

The HADB stores history information about the service usage which is needed when sending evidence to the TTP. Though the RADIUS protocol is stateless, the NoRSU usage requires keeping track of the state, so decision about when to accept or deny the incoming connection based on the RADIUS authorization and the accounting information can be made. Since the host association database is removed when a HIP connection is closed, the connection related information must be stored before sending CLOSE/CLOSE ACK packets.

6.2 Packet Size Restrictions

To avoid vulnerability against DoS attacks, packet sizes were decided to keep under the IPv6 recommended MTU size of 1280-byte to prevent fragmentation of the packets [43]. Table 6.*4* shows mandatory HIP control packet sizes with default sizes of 1536-bit D-H key length and 1024-bit RSA keys in the signatures. The two possible sizes for R1 are offer certificate and offer certificate along TTP certificate. The three possible sizes for the I2 packet are response certificate, response and TTP certificate and response certificate with TTP and delegation certificates.

Table 6.4. *HIP packet sizes in bytes.*

Encoding	**I1**	**R1**	**I2**	**R2**
S-enc.	152	966/1352	968/1354/1698	216
Eff.	152	807/950	807/950/1137	216
Normal	40	640	608	216

6.3 Certificate Modifications

Vanilla HIP has support for handling X.509v3 and SPKI based certificates with a minimal required content. This support was extended with NoRSU certificates. Figure 6.1 shows certificate parameter structure [44]. The only restriction for the certificate field length is the size of the control packet.

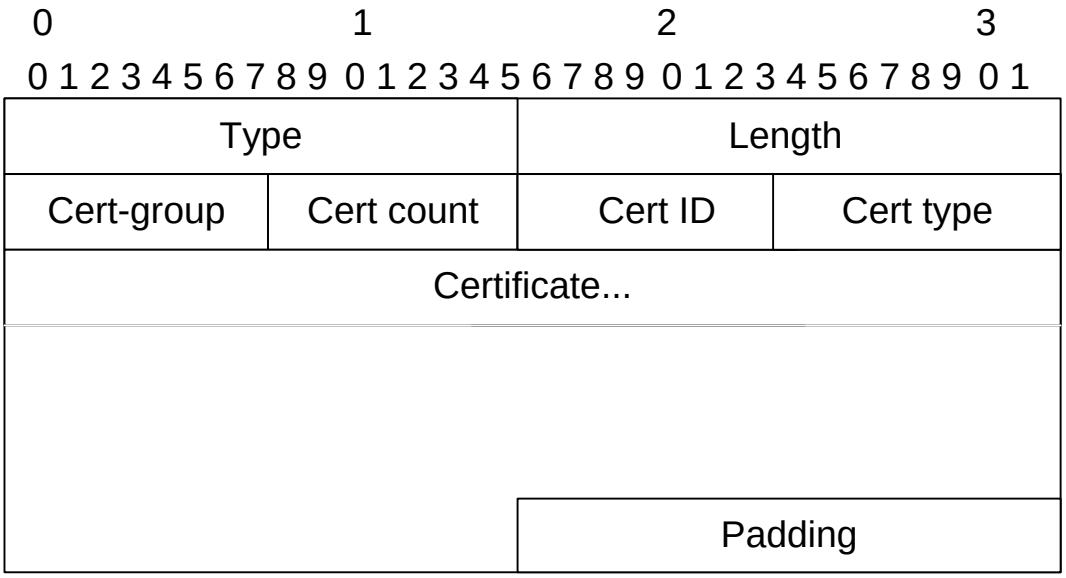

Figure 6.1. *HIP Certificate parameter structure.*

Certificates are created based on the configuration file. The parser reads this configuration file and transforms configuration options to SPKI statements. Options are in a simple attribute value-pair format. Options for NoRSU certificates were added to this configuration and parser was modified to accept new options. Table 6.5 shows new certificate types, HIP parameters and numbers. HIP allows grouping of CERT parameters so long certificates can be divided into multiple control packets. This option could not be used in this solution since the agreement of the service usage must be made during the base exchange.

Table 6.5. *New certificate parameters.*

Certificate	**HIP Parameter**
SPKI Encoded Offer Certificate	HIP_PARAM_CERT_SPKI_OFFER (32826)
SPKI Encoded Response Certificate	HIP_PARAM_CERT_SPKI_RESPONSE (32827)
Efficient Encoded Offer Certificate	HIP_PARAM_CERT_EFF_OFFER (32828)
Efficient Encoded Response Certificate	HIP_PARAM_CERT_EFF_RESPONSE (32829)

For the efficient encoding, the same configuration file parser was used for the options. Options were transformed into binary encoding. Table 6.6 describes used SPKI statements in the efficient encoding format. All statements are encoded in a type value format containing fixed size values to avoid collisions when parsing the certificate.

Table 6.6. *SPKI statements and binary encoded equivalents.*

SPKI statement	**Binary encoding**	**Size in bytes**	**Description**
cert	0xaeba	4	Beginning statement for SPKI certificate
validity not after	0xff02	6	Validity period of the offer
issuer	0xff03	22	Issuer of the certificate
offer s	0xff04	6	Offer using time based charging, unit second
offer b	0xff05	6	Offer using volume based charging, unit byte
hash of offer	0xff08	22	Hash of the offer
hash chain anchor	0xff09	22	Anchor value of the used hash chains
subject	0xff10	22	Subject of the certificate
target service	0xff11	22	Target service for delegation
amount-max	0xff12	6	Maximum amount of usage, unit second
propagate	0xff13	2	Is propagation allowed
validity period	0xff14	10	Validity range of the certificate
signature-rsa-sha1	0xffaa	128	RSA-SHA1 signature
signature-dsa-2048	0xffbb	448	2048 bit DSA signature

In the SPKI encoded certificate verification, a parser based on regular expressions is used. Based on the input statement, the parser returns value of the statement. For NoRSU statements the parser was modified to verify NoRSU related values. The parser also

configures the NoRSU connection attributes in the HADB based on the parsed values. Efficient encoding is parsed based on known lengths of the fields.

Figure 6.2 shows example certificate in the efficient encoding format presented in hexadecimals and in the canonical S-expression formatted for readability. The efficiently encoded SPKI statements are bolded.

Canonical S-expression:

```
(
(4:cert
(6:issuer(4:hash4:sha120:abcd..))
(5:offer(4:time(1:1(3:s60)))
(8:validity(9:not-after19:2012-01-16_19:00:00))
)
(9:signature(8:rsa-sha1128:abcd...))
)
```

Efficient encoding:

03 00 00 c0 01 01 00 32 f0 b2 48 b7 **ae ba ff 10** d4 33 61 21 fb 6a
64 e5 ad 5a 8a 9c 4a 97 69 73 c3 47 a0 2e **ff 04** 00 03 **ff 02** 4e 34
3f 2e **ff aa** 3d c6 7b 62 c6 b3 e9 21 df 20 e0 85 d8 c0 a5 17 18 77
59 4a ed d5 be b6 75 e0 63 e3 1b 57 ec 16 2b 06 d7 43 14 ab f1 be
23 f0 9f c5 10 a1 fc ce 8d 24 7c 0b a8 a5 3f db 63 70 65 07 95 21
b2 14 7d 84 be d1 cd 3f c2 4c 40 e3 1b a6 1c 21 b2 cd 0b b8 b7 04
dd 17 f7 3c d5 b3 c3 c6 b1 fe 44 67 65 09 f7 05 ba 4c 93 9b 6b 2f
8c 9a a1 d6 0b 02 3a 29 ce 9d ab 8e e0 e6 fd dd e5 01 b7 e2 3f 09
20 01 00 10 ad e5 28 6d 36 84 7d e7 91 80 a7 f6
00 00 00 00

Figure 6.2. *Offer certificate in efficient and canonical S-expression encoding format.*

6.4 Hash Chain Support

For testing purposes the hash chain method is capable of creating two kinds of hash chains. The traditional chains are created using SHA-1 hash function and a random seed value. Parameters related to the chain are located in the HIP configuration file. These parameters are the hash chain length and the amount of chains. The first chain is created after receiving the offer from the server. The mechanism supports also importing externally created chains. The used hash chain values are stored in the host association database. This enables the implementation to recover from the situation where several hash chain pieces are lost in the transport channel.

It is also possible to chain multiple traditional hash chains using the hash chain switching. In the hash chain switch a special switch parameter is created. The switch parameter contains a concatenation of the second last value of the exhausted chain and the anchor value of the new chain. This concatenation parameter is signed with client's private

key to ensure the continuation of the evidence chain needed in the non-repudiation. If a UPDATE packet containing the switch parameter is lost, it is impossible to recover from the situation and the connection must be terminated.

The second kind of hash chains are infinite length hash chains, which are created using RSA algorithm and host's private key. The first value is created from a random seed value and the next element is generated using previous value as a seed. Because of the packet size caused by infinite length chain pieces, this functionality is disabled by default. It can be enabled from the HIP configuration file or using the HIP configuration tool. Appendix A: Example Norsu Configuration shows an example configuration file.

6.5 Charging

Two types of charging schemes were implemented: time and volume based charging. In the time based charging the customer pays for the time he or she uses the service. In the volume based charging the customer pays for some resource consumed during the service. In this solution the chargeable resource is transferred data. Other chargeable resource could be f. ex. processor time or amount of database queries.

6.5.1 Time Based

The time based charging uses ticks to determine the moment to send the hash chain piece. The tick length is in the offer certificate and when the client accepts the offer; both the entities adjust their internal timers to the agreed tick length. Any time synchronization protocol is not used so both timers run in a little bit different time. Timers coarsely synchronize when the hash chain piece is sent and verified. Depending on whether the timer reset is made on UPDATE sent or UPDATE ACK, server's or client's clock is always a little bit ahead. In this solution client's timer is a little bit ahead to avoid unnecessary timer expirations caused by the transmission delay. Server's timer contains a threshold which allows the delay of three ticks before the timer expires. Because of coarse timing, extremely long or short tick times should not be used to achieve smooth operation. In this solution the default tick length is one second.

6.5.2 Volume Based

Transferred data based charging in this solution is implemented using HIP Firewall. The used threshold value is in the offer certificate and when the agreement is made, the client and the server setup their HIP Firewalls with the agreed value. The used socket options are described in Table 6.7. The client monitors incoming IPsec traffic for the specific connection and when the threshold value is exceeded, HIP Firewall signals HIP Daemon to send the hash chain piece. The server monitors outgoing IPsec traffic for the specific connection. Figure 6.3 shows interaction between HIP Daemon and Firewall when a traffic threshold is first set and later exceeded.

Table 6.7. *Socket options used in firewall communication.*

Signaled event	HIP Socket option
Threshold exceeded	SO_HIP_HASHCHAIN_OFF (203)
Reset traffic threshold monitor	SO_HIP_COUNTER_RESET (207)
Set value to traffic threshold monitor	SO_HIP_COUNTER_SET (208)

If the hash chain piece is not received when the threshold value is exceeded, depending on how many missing chain pieces is allowed, the connection expires. Since IPsec traffic is encrypted, also TCP/UDP and IP overhead is included in the monitored traffic. This causes inaccuracy to the measurement.

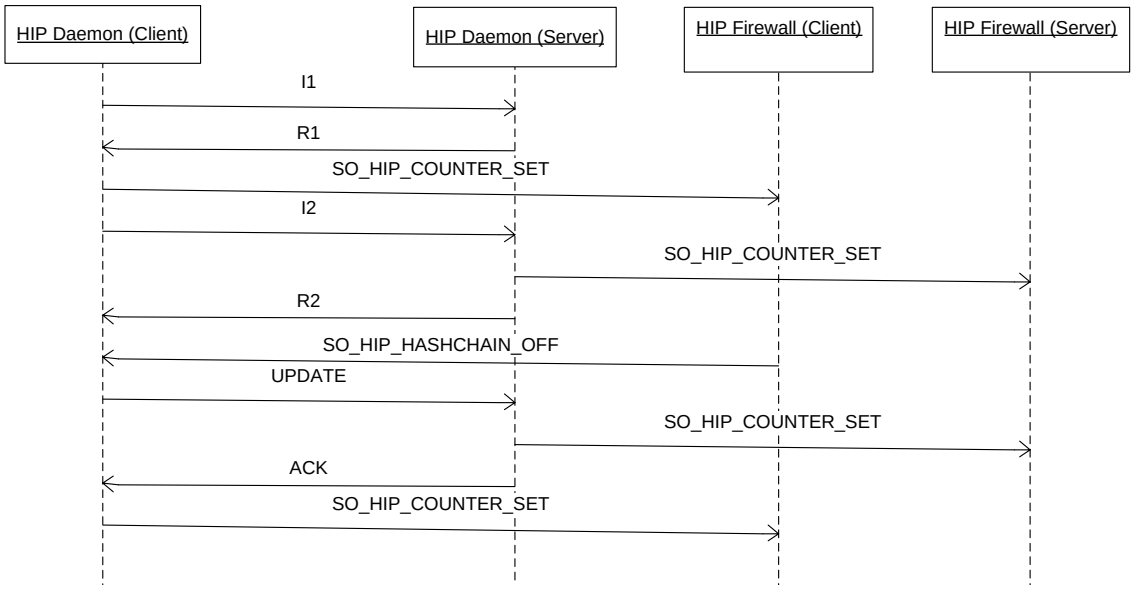

Figure 6.3. *Interaction between HIP Daemon and Firewall.*

HIP Firewall is based on iptables tool [45]. HIP Firewall monitors incoming and outgoing packets based on preconfigured rules. Because of this, settings for the HIP Firewall must be configured before the base exchange. This restricts the usability since HITs, IP address, or the used network interface must be known beforehand to create the appropriate rules for the connection monitoring.

6.6 HIPconf

HIPconf is a utility to set the runtime configuration to HIP Daemon. HIPconf uses the internal communication mechanism and sends parameters and configuration options to HIP Daemon as socket options. HIPconf works one way only, so it does not receive a response from the HIP Daemon for the sent commands. HIPconf is used to enable and enable various NoRSU related options. The possible options are listed in Table 6.8. All existing configuration options remained untouched.

Table 6.8. *Added HIPconf options.*

Configuration option	**HIPconf parameter**	**HIP socket option (number)**
Enable NoRSU service	norsu on	SO_HIP_NORSU_ON (204)
Disable NoRSU service	norsu off	SO_HIP_NORSU_OFF (205)
Enable NoRSU server mode	norsu server on	SO_HIP_NORSU_SERVER_ON (209)
Disable NoRSU server mode	norsu server off	SO_HIP_NORSU_SERVER_OFF (210)
Enable SPKI encoding	encoding spki	SO_HIP_NORSU_ENCODING_SPKI (211)
Enable efficient encoding	encoding eff	SO_HIP_NORSU_ENCODING_EFF (212)
Enable infinite length hash chains	pubkeyhash on	SO_HIP_NORSU_PUBKEYHASH_ON (213)
Disable infinite length hash chains	pubkeyhash off	SO_HIP_NORSU_PUBKEYHASH_OFF (214)
Recreate pre created certificates	create offer on	SO_HIP_NORSU_CREATE_OFFER (215)
Enable RADIUS message relay to external RADIUS server	radrelay on	SO_HIP_RADIUS_RELAY_ON (219)
Disable RADIUS message relay to external RADIUS server	radrelay off	SO_HIP_RADIUS_RELAY_ON (220)

All of the configuration options cannot be used simultaneously. Certificate encoding can be either SPKI or efficient encoding for example. The encoding mode must be chosen before the base exchange and cannot be changed on the fly during the connection.

6.7 RADIUS Implementation

The RADIUS support is divided into two components. RClient offers functionality to use the RADIUS client with the HIP internal communication mechanism. RDaemon offers functionality to establish a HIP connection with TTP and RADIUS message encapsulation to HIP control packets. Figure 6.4 shows interaction between different components. Components and phases are similar to Figure 5.1.

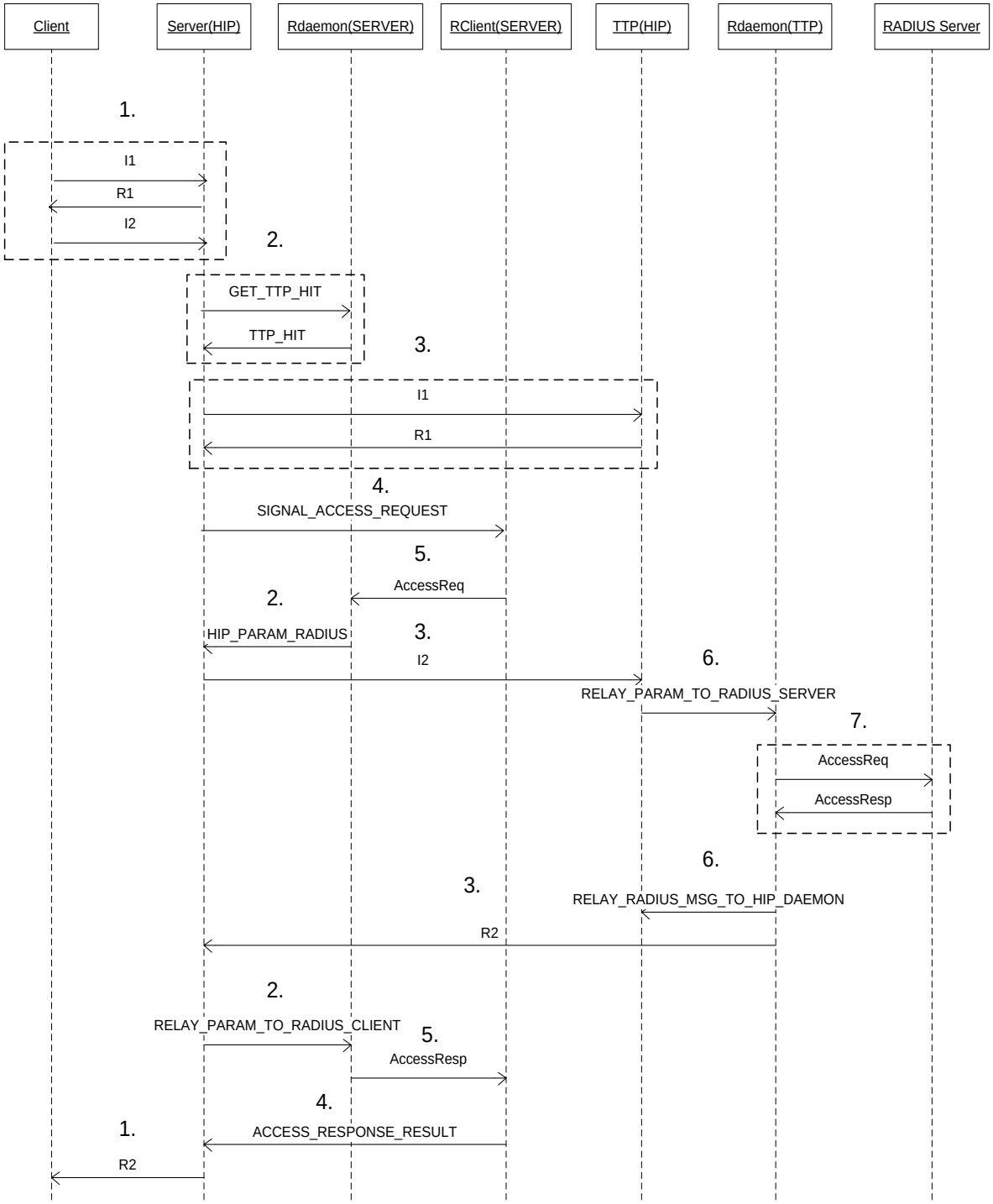

Figure 6.4. *Interaction between different components.[41]*

If the HIP connection between the TTP and the server is already established when the base exchange between the client and the server begins, RADIUS messages are delivered to the TTP using UPDATE packets in the phase 3.

.

6.7.1 RClient

RClient is a daemon which performs the RADIUS protocol client functionality. RClient offers an API to RADIUS clients and HIP internal communication mechanism. RClient receives RADIUS connection requests from HIP Daemon and is connected to RDaemon. When HIP Daemon requests a RADIUS message, RClient creates an appropriate message using attached the RADIUS client. After this the RADIUS client sends the message to RDaemon which emulates the RADIUS server. The RADIUS message travels through the HIP connection to the actual RADIUS server. HIP socket options used between RClient and HIP Daemon are described in Table 6.9.

Table 6.9. *RClient communication socket options.*

Signaled event	**HIP Socket option**
Access Request	SO_HIP_RADIUS_SEND (222)
Accounting Request (Start)	SO_HIP_RADIUS_ACC (223)
Access Accepted	SO_HIP_RAD_AUTH_OK (224)
Accounting Request (Stop)	SO_HIP_RAD_ACC_DATA (225)

When the response is received, RDaemon returns the response to RClient. This functionality hides the actual transport channel from the RADIUS client. The RADIUS client interprets the contents of the message and signals HIP Daemon about the response type. RClient supports access and accounting request messages. RClient contains a parser to create RADIUS vendor specific attributes based on the parameter received from HIP Daemon. Figure 6.5 shows interaction between components in connection teardown. HIP Daemon sends evidence collected in the HADB during the connection to RClient. Certificates are transported in HIP_PARAM_CERT and hash chain sets in HIP_PARAM_RADIUS_HASHVALUE. RClient creates vendor specific attributes based on the received evidence. Table 6.10 shows how evidence is encapsuled in VSA.

Table 6.10. *Evidence in Vendor Specific Attributes.*

Archived evidence	**Vendor Type**
Offer certificate	1
Response certificate	2
TTP certificate	3
Delegation certificate	4
Total received hash chain pieces	5
Hash chain anchor value	6
Last received chain value	7
Received pieces in chain	8
Hash chain switch value	9

If the connection to TTP is already established when closing procedure begins, the UPDATE packet is used to carry RADIUS parameters. Otherwise connection to the TTP is established to deliver the evidence. One UPDATE packet is able to carry Offer, Response, TTP, Delegation certificates and two hash chain sets, if the certificates use the efficient encoding. For rest of the hash chain sets as much UPDATE packets as needed is sent to deliver all connection related data to the TTP.

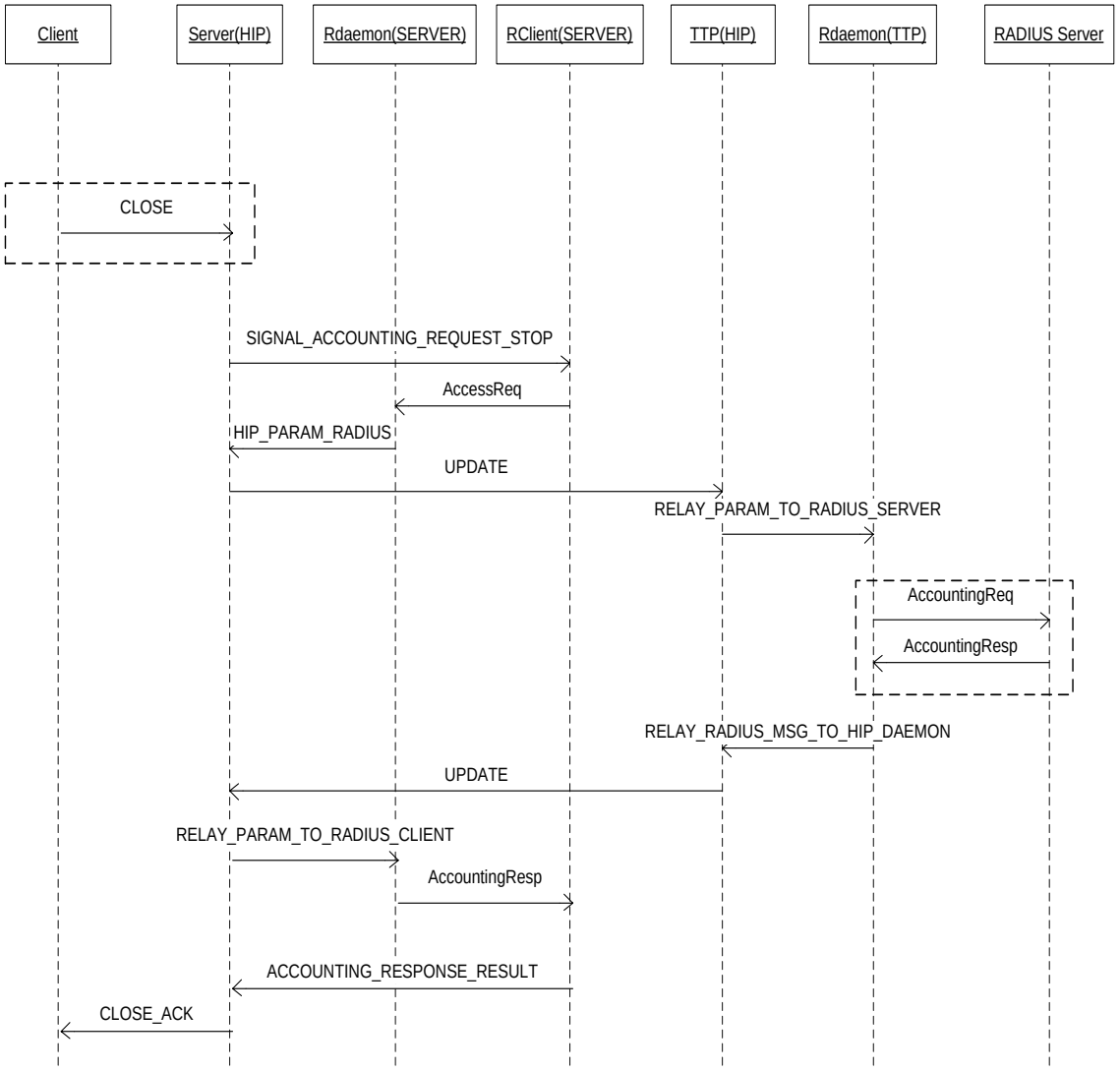

Figure 6.5. *Connection teardown.*

6.7.2 RDaemon

RDaemon implements two major functionalities. The first one is TTP finding mechanism. Available TTP HIT's are configured to the HIP configuration file under [ttp_hit] section. HIP Daemon uses internal HIP messaging via dedicated socket to signal TTP

connection request to RDaemon. The same mechanism is used in the communication between HIP Daemon and HIP Firewall.

When RDaemon receives a TTP connection request, it chooses the first suitable HIT and triggers HIP a base exchange via internal communication mechanism. Because HIP Daemon runs in a single thread, all other active and incoming HIP connections are blocked until the triggered base exchange is finished.

The second functionality is RADIUS message encapsulation and decapsulation. HIP Daemon works in cooperation with RClient. RDaemon listens to the port 985 for UDP messages and emulates a RADIUS server to RClient. After a RADIUS message is received RDaemon creates HIP Parameter from the received RADIUS message and uses internal HIP communication mechanism to deliver the parameter to HIP Daemon. After this, HIP Daemon applies the received parameter to an appropriate HIP control packet. Table 6.11 shows mapping between RADIUS messages and HIP parameters.

Table 6.11. *Mapping between RADIUS messages and HIP Parameters.*

RADIUS Message	**HIP Parameter**
Access-Request	HIP_PARAM_RADIUS (32832)
Accounting-Request	HIP_PARAM_RADIUS_ACC (32835)

On the other side, HIP Daemon delivers received HIP Parameters containing RADIUS messages to RDaemon. RDaemon extracts the RADIUS message and forwards it to the receiver. Depending on the role, a server or a client, the receiver can be a RADIUS server or RClient. RDaemon remains listening to the input channel for the possible response message.

7 ANALYSIS

The performance of the implementation was tested to study the effects of the non-repudiation modifications. The other focus of testing was to evaluate the fulfillment of functional and non-functional requirements introduced in Chapter 4. Test setup was designed to act as a typical usage scenario for the implementation. Actual test cases often used in software engineering [46] were not designed. Since the focus of the work was to evaluate the user experience and the functionality of the system as a whole, it was found not so useful to put effort to verify the operation of some minor feature. Confirming the requirements guided the testing process.

7.1 Test Environment

Testing was performed in a laboratory environment. The test setup contained three machines which were connected to each other. Machines were connected to a local area network. In the RADIUS roaming setup introduced in 0, 1000Mbit Internet connection to FUNET was used. Figure 7.1 shows the physical setup in the laboratory. NoRSU1 and NoRSU2 were connected using SMC Wireless 54Mbit USB Adapters. NoRSU2 and NoRSU3 were connected with 1000Mbit Ethernet connection.

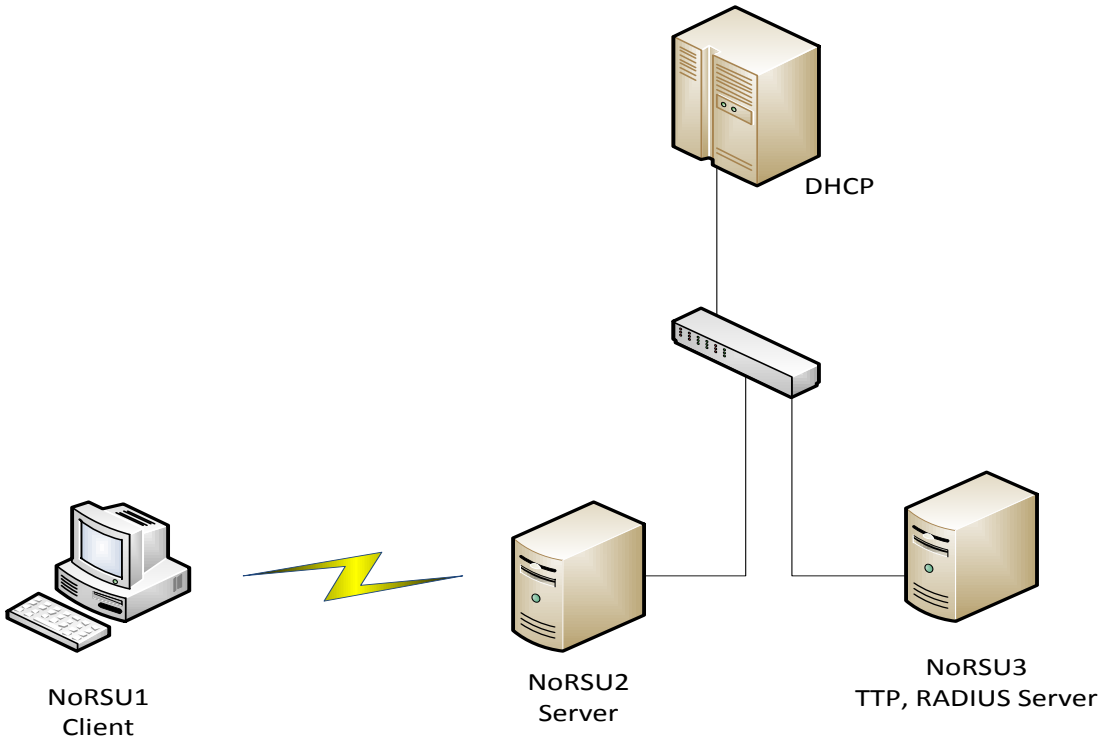

Figure 7.1. *Physical setup in the laboratory.*

DHCP server was running in a VMWare based virtual machine. IPv6 addressing was used between all hosts and IPv4 addressing was disabled.

7.1.1 Test Platform

The used test machines had corresponding performance of today's low end home computers. Tests were also performed in a virtual machine environment, but results were inconsistent because of the unbalanced processor load. The virtualization platform machine had two physical processor cores, but when three virtual machines and host operating system were running at the same time, the load grew too high. Machines introduced in Table 7.1 gave consistent results through the measurements.

Table 7.1. *Test machine specifications.*

Device	**Processor**	**Memory**
NoRSU1	Pentium 4 1,7 GHz	1024MB
NoRSU2	Athlon XP 1,53 GHz	1024MB
NoRSU3	Pentium 4 2,4 GHz	1024MB

Because of the physical setup instead of virtualization, testing was restricted to single connections between hosts. Because of the different processor types, processor loads were not monitored. Hyperthreading in NoRSU3 was turned off during the tests.

7.1.2 Test Software and Execution

The operating system in the test machines was Debian 5.0.2 "Lenny". The kernel version was 2.6.26-2-686, which contains BEET functionality. All tests were performed using 1024-bit RSA keys and 1536-bit Diffie-Hellman group. For other options HIP default values were used. A special version of HIPL 1.04 was used, which contains possibility to set measurement points in the source code. These points were used to measure different events during the base exchange. The original source code contained approximately 50 000 and non-repudiation modifications consisted of 5000 lines of code. This includes new and modified lines.

7.2 Performance Measurements

In the performance measurements, the main focus was in the base exchange duration. It can be used to evaluate how well the implementation would work in the actual scenarios. It also gives some hints about how many clients one server can handle. If one base exchange takes some amount of time, this time can be used to evaluate the user experience in the client side. All measurement results are comparable with each other.

7.2.1 Non-Repudiation Modifications

The first round of measurements was performed with non-repudiation modifications. In this setup, NoRSU1 and NoRSU2 were used. Figure 7.2 shows base exchange duration on the server side in three cases. Normal refers to Vanilla HIPL implementation and two others to the implemented version. Measurements were made using a script which started a base exchange, waited 4 seconds, disconnected and started the base exchange again after 1 second. Results are averages of 100 measurements. With all versions measurement began on the server side, when server started to process the I1 packet and finished after the R2 packet was sent.

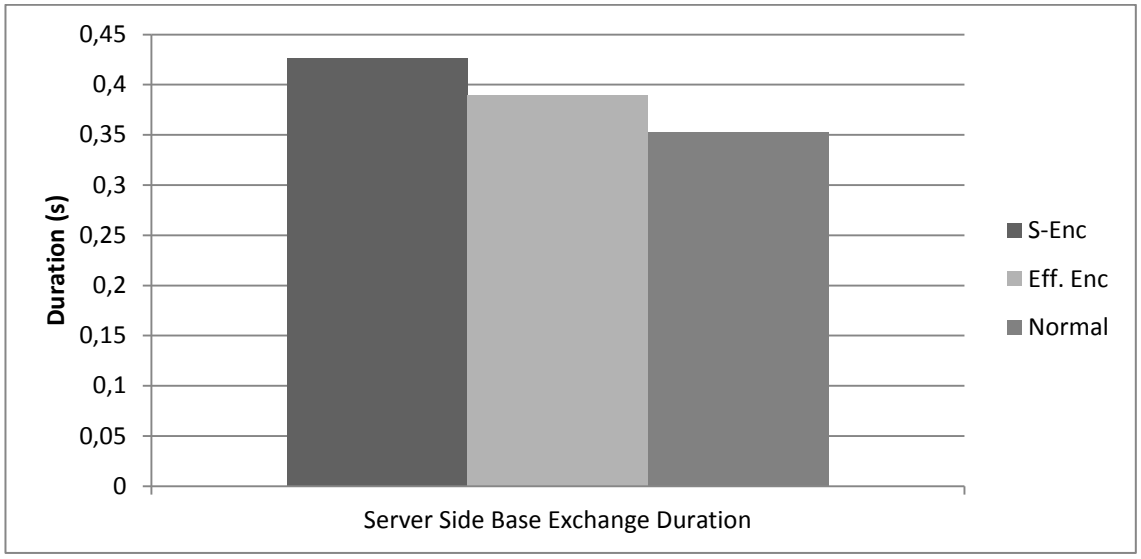

Figure 7.2. *Server side base exchange duration.*

In the version with NoRSU modifications, the verification of the response certificate was done on the client side. The creation of the offer certificate was not counted since it is pre-created when creating a R1 packet. Figure 7.3 shows the base exchange duration in same three scenarios on the client side.

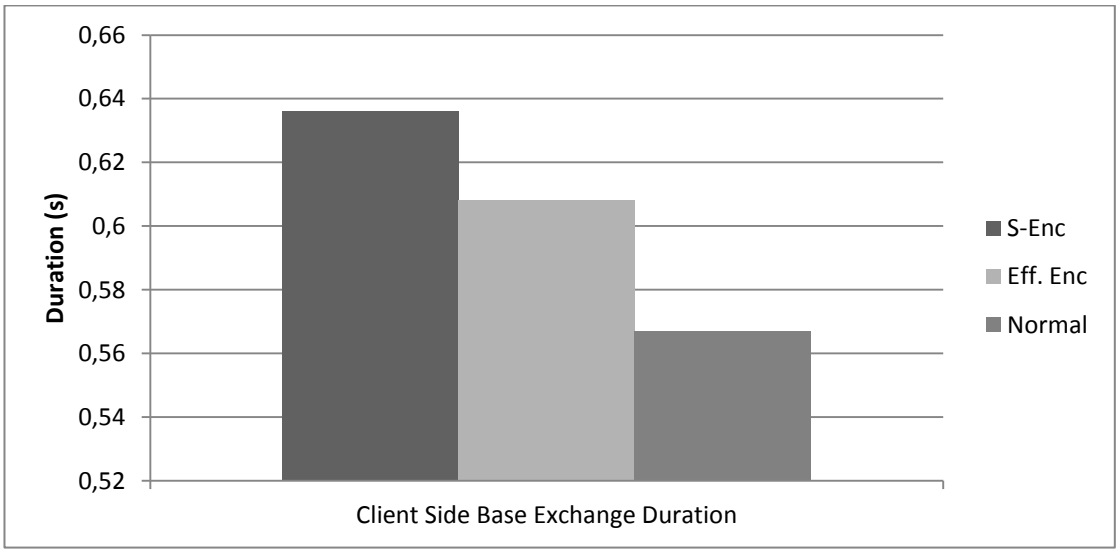

Figure 7.3. *Client side base exchange duration.*

On the client side the measurement was done from the I1 packet creation to the verification of the R2 packet. This includes the verification of the offer certificate and creation of the response certificate. Traditional, not infinite length, hash chain anchor value was used in the response certificate. The hash chain was created during the creation of the I1 packet. The hash chain length of 100 was used. Table 7.2 shows processing times in more detail. Results were quite near what was expected. The most of the delay in the server side is caused from the verification of the response certificate RSA signature. In the client side verification of the offer certificate signature and creation of the response certificate signature causes the delay.

Table 7.2. *Detailed processing times in seconds.*

	R1 Handling (Client)	**I2 Handling (Server)**	**I2 Creation (Client)**
S-Enc	0,106	0,116	0,039
Eff. Enc	0,090	0,093	0,036
Normal	0,062	0,085	0,017

The difference between the S-Encoding and efficient encoding is caused by the parser. The S-Encoded parser is based on the regular expressions and allows more flexible certificates. The efficient encoding is assumed to follow strict order in the certificate structure which improves the performance but weakens the flexibility. Figure 7.4 shows performance in the creation of the UPDATE packet.

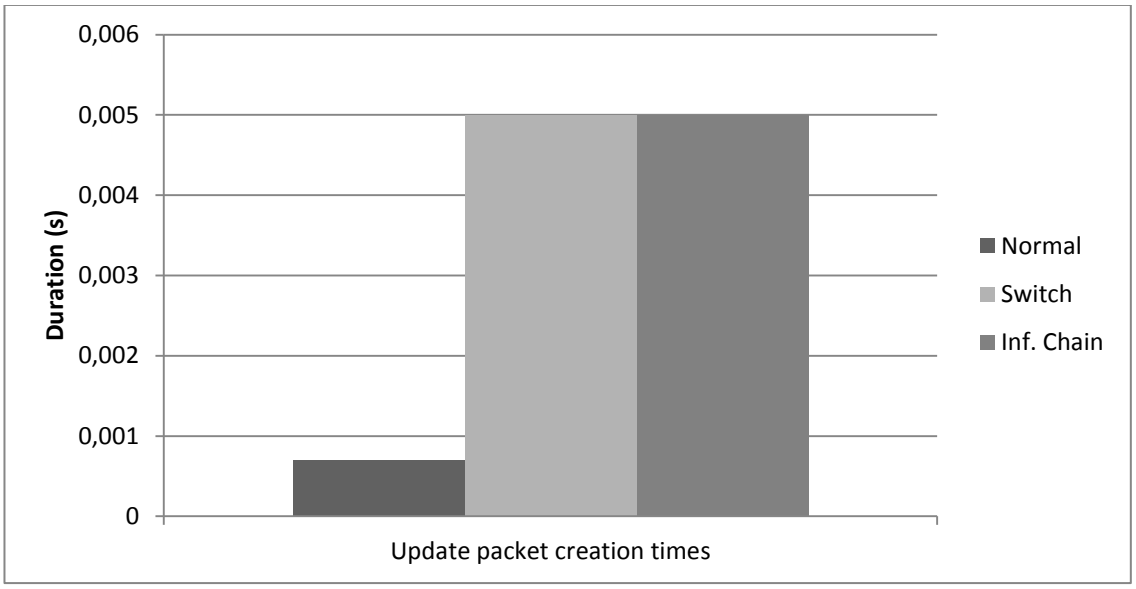

Figure 7.4. *Client side UPDATE packet creation times.*

Vanilla HIP UPDATE packet contains RSA signature. However, in this implementation it was removed as unnecessary since hash chain mechanism is able to provide authenticity of the packet. Normal column in Figure 7.4 and Figure 7.5 means modified UPDATE packet, which contains a hash chain piece. UPDATE packet containing the

switch parameter is used to replace the exhausted chain to a new one. It also binds chains together with the client RSA signature, which causes the greater time consumption. Similarly infinite hash chains are based on signatures. Even though both times are approximately five times higher than normal UPDATE packet creation times, they are still nearly ten times faster than the creation of the I2 packet.

The most time consuming operation on the processing of the switch parameter and the infinite length hash chain is RSA signature verification. Difference between switch parameter and infinite length chain verification time is caused by the verification of the traditional chain piece. Figure 7.5 shows UPDATE packet verification times in the server side, which is evaluated in three cases.

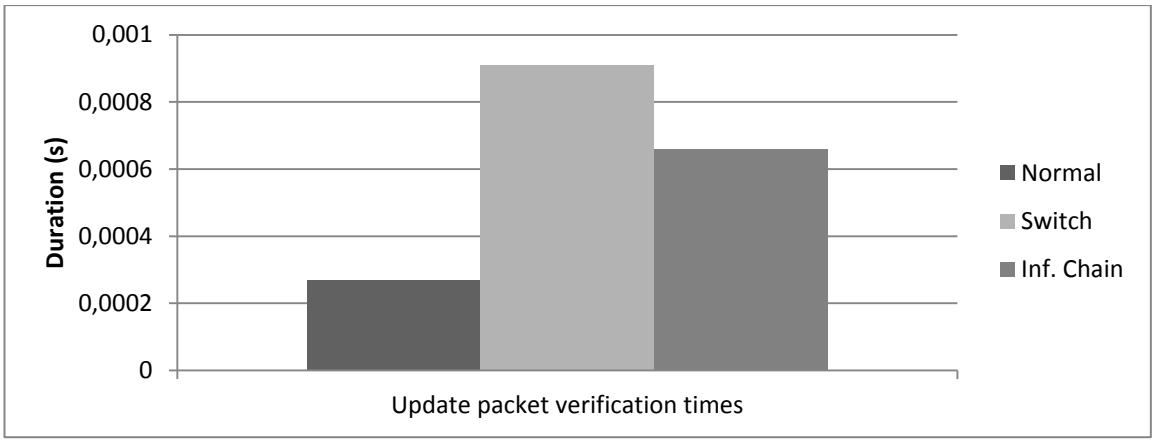

Figure 7.5. *Server side UPDATE packet verification times.*

Again, normal refers to the packet with a single hash chain piece. Since the switch parameter contains values from two chains, it must be verified that other value belongs to the old chain before taking the new chain's anchor into use. Though these verification times are much faster than any base exchange related operations, it must be noticed that the server typically receives messages from multiple customers. However, it is unlikely that many clients send switch parameters simultaneously. More typical operation to the server is the verification of the traditional chain piece, which takes only 0,0002s.

7.2.2 RADIUS Support and Roaming

The next measurement target was the base exchange duration when using TTP. In the first setup TTP connection was static and during the base exchange the server verified the client's identity from the TTP using the RADIUS protocol, which was encapsulated inside HIP UPDATE packets. In the second setup, a connection between the server and the TTP was dynamic during the base exchange. Figure 7.6 shows the base exchange duration in the client side when using TTP.

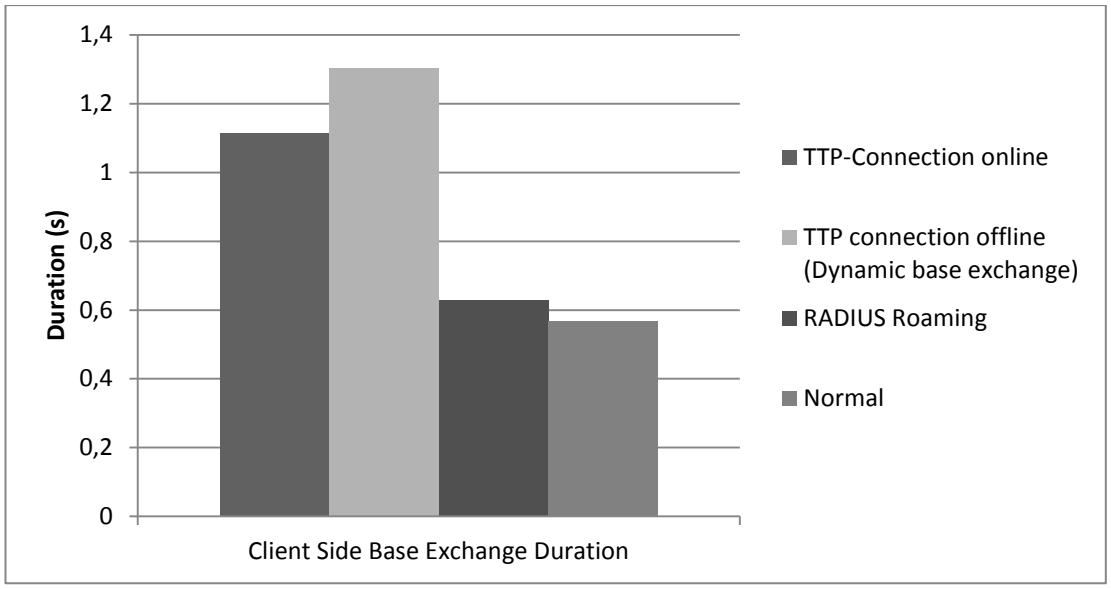

Figure 7.6. *Base exchange duration when using TTP.*

The use of dynamic base exchange duration is almost three times greater in the client side than the normal HIP base exchange. Typically, this happens only once and the connection to the TTP remains open. The most of the duration is caused by slow socket handling when using HIP internal communication mechanism between daemons. Reason for this was artificial delay in socket handling. Decreasing this delay caused errors in the socket functionality, especially in the UNIX *select()* function performance [47]. Although this issue was investigated, no obvious reason was found. It must also be noticed that because of the single threaded nature of HIP Daemon, it is in the blocking state for the whole base exchange duration, which can be problematic with a large amount of clients. In this setup the RADIUS server and TTP located physically in the same machine, but in a real life scenario they can operate in separate machines which causes even more delay.

RADIUS roaming was tested in an additional test setup. NoRSU1 acted as a client who wished to roam into the network where NoRSU2 acted as a gateway. During the base exchange between NoRSU1 and NoRSU2, NoRSU2 asked confirmation from the FUNET RADIUS server, if NoRSU1 is allowed to roam. This server acted as a proxy and forwarded the request to the RADIUS server of the home organization of NoRSU1, which made decision to accept or decline the request. This server replied to the FUNET server which relayed the answer to NoRSU2. If roaming was allowed, NoRSU2 continued the base exchange. Figure 7.7 shows the test setup.

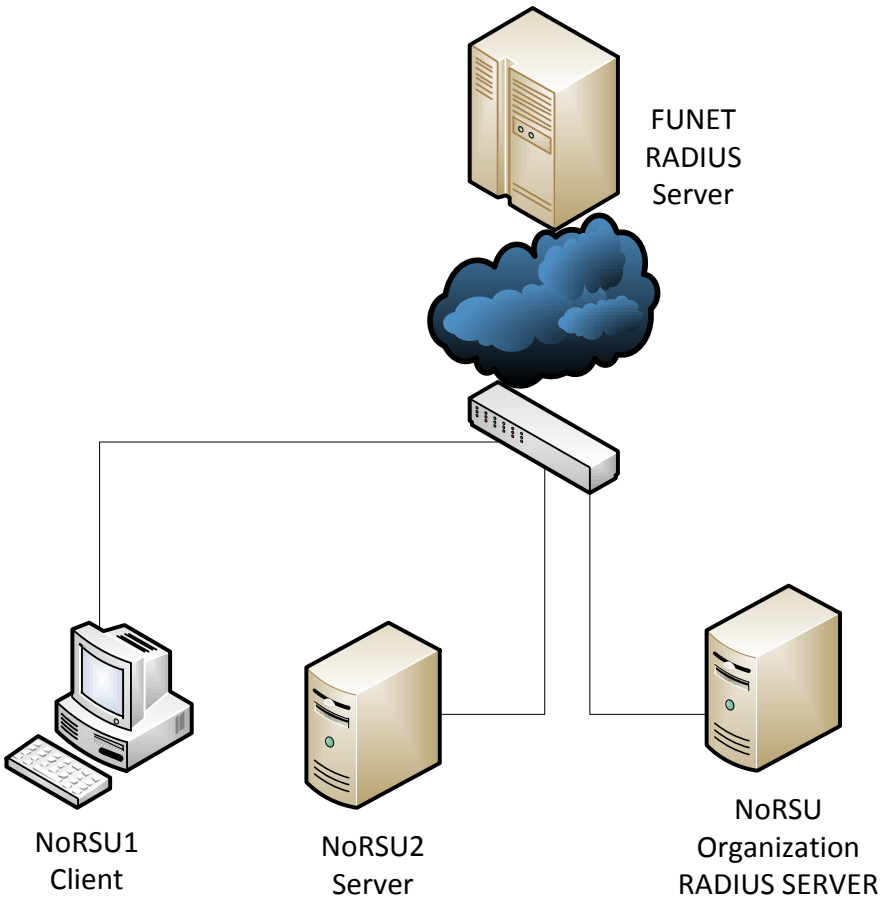

Figure 7.7. *Test setup in the RADIUS roaming scenario.*

In this setup RADIUS messages went directly from NoRSU2 to the FUNET server through the Internet. Because there is no delay caused from the HIP socket handling and processing, roaming is almost as fast as the normal HIP base exchange. However, the connection between NoRSU2 and the FUNET RADIUS server is not protected, which makes this scenario not feasible to actual use, since privacy of the roaming client is compromised.

7.3 Implementation Analysis

The implementation was evaluated against the non-functional requirements which were set in Section 4.2. Objective measurements to these requirements are difficult to perform, so the main focus is on introducing the solutions which were made in the implementation. In overall the requirements were pretty loose due to the prototype nature of the project.

7.3.1 Expandability

The implementation is pretty modular in most core functions. New certificate types can be added by just implementing new creation and verification functions and adding new parameter types. New encoding types for existing certificates require re-implementation of existing creation functions. Since various encoding types exist, creating a general

encoding framework was too time-consuming within the confines of the project. New cryptographic hash functions can be easily taken into use by implementing hash chain creation and verification functions for new hash function type. However, new hash functions must not produce larger chain pieces than the existing one, because of the packet size restrictions in the Section 0.

The configuration mechanism follows the original HIPL implementation, so new configuration definitions must follow the HIPL mechanism. This includes creating a parser for the configuration option and adding new socket options for the configuration. The used AAA-protocol can be easily changed by creating a new AAA Daemon which offers message handling for Access/Accounting requests and responses. The same daemon can offer improved evidence handling and a database client. Support for different kind of service offer setups requires modification to several modules. Offering a general platform for unknown service types turned out to be difficult with the existing architecture. One of the main reasons is the single threaded functionality of HIPL. This caused that every action done during the base exchange stalled the other connections until the action was finished.

7.3.2 Compatibility

Tests revealed that the implementation is able to interoperate with the original HIPL. Actual version control between different versions of NoRSU implementations was not implemented. Future modifications in the server or the client side may or may not break the compatibility with original HIPL. Because there is not any general handling functionality for parameters, HIPL assumes that parameters are constructed strictly under the definition of [8]. As a whole this implementation provides the minimum operations to realize NoRSU functionality. Transparency to the upper layer protocols and middleboxes was maintained.

7.3.3 Usability

Because of the subjective nature of usability, objective evaluation is difficult. The name value pair type configuration was implemented, but less Linux oriented users may find it difficult to use. Since this implementation aims to support various kinds of service offerings, a typical user may not be very skilled in Linux. Still, most of the users find it very welcome to get the possibility to reduce the risk of being defrauded. Because the implementation is command line based, information about the connection is provided as text based. Understanding this information may be difficult to an average user. However, the whole configuration can be done in a single file, so the operator could offer these configuration files including pre-created hash chains but this can compromise the non-repudiation. Pre-created hash chain is already known by the creator, so it can forge the amount of sent pieces. The only configuration task for the user would be putting the configuration file to the right place and configuring right key pair for the device. Even

though this may be considered complicated, the whole functionality of the implementation should be transparent to the user in normal situations. When an upper level application requires a connection which requires non-repudiation, HIP base exchange of the implementation is triggered automatically. Handling the error situation may require some expertise from the user.

7.3.4 Security and Non-Repudiation

All the existing HIP security properties were maintained. Yet, there are multiple threats against NoRSU. Identity faking is protected with the use of Host Identities. However, the Host Identity binds the identity only to the device, not to the actual user. This must be taken into account when planning the service scenario. The offer certificate binds the service provider to the offer and the response certificate binds the client to the agreement. Both certificates are signed with host's private key, which protects certificates against forging. The response certificate contains the hash of the offer which prevents the client from tampering the offer to a more favorable one. TTP certificate provides additional trust for the service provider about the customer. The hash chain anchor is bound to the signed certificate. The payment is made by sending hash chain pieces. Due to the irreversible nature of the hash chain, only the chain creator knows the predecessor of the chain piece. Now payments are bound to payer's identity and identity is bound to the agreement. So the client cannot deny payments afterwards. Messages which contain hash chain pieces are encrypted, so even if the message was captured, the capturer could not gain advantage of it. If the client stops sending the chain pieces, the service provider stops the service. If a threshold against missing chain pieces is used, a dishonest client could get advantage since service is not stopped immediately after a chain piece is missing. The implementation cannot protect against this directly, but this can be avoided by keeping the single chain piece value minimal and keeping the chain sending interval rather small.

If the client does not receive satisfactory service, it can simply stop sending chain pieces. In certain service types, this may cause a problem. When using NoRSU based streaming, the client may face unsatisfactory service in the middle of a movie. Now the client has already paid half, but even though he or she does not have to pay further, value of a half seen movie cannot probably compensate the price the client has already paid.

7.3.5 Feasibility

The implementation works pretty well in a prototype usage. Using the implementation in a real environment requires some further work. The implementation offers a technical platform for the NoRSU, but lacks non-technical procedures. For example, the situation, where evidence is needed to point out a dishonest participant, needs some kind of defined procedure and an adjudicator. The hash chain delivery and key management re-

quires also some business planning. Even though the Host Identity helps to provide identity for a device, there must be also binding to the device owner's identity. This user identity management is out of the scope in this implementation. The implementation technical side also requires some fine tuning depending on the usage scenario. For example, used RADIUS parameters may require adjustments to acquire compatibility with the deployment environment.

One of the biggest feasibility problems is the single threaded functionality of HIPL. This makes the handling of large user amounts very difficult. Depending on the speed of the authentication, users will face delay. Also malicious clients may stall the base exchange, which causes even more delay to various operations for other users. To prevent this, timeouts in the base exchange must be minimized which may cause problems to users with a large network delay. Delay in the network may cause problems in the timers when using time based charging.

8 FUTURE WORK

Before the implementation began, HIPL was chosen to act as a platform. HIPL is the most actively developed and supported HIP implementation at the moment. As earlier mentioned, one of the biggest downsides in the NoRSU implementation was the single threaded design of the HIPL. Since the development of the platform was too time consuming to this project, the restriction had to be accepted. In the future, HIPL could be modified to support event based functionality, where operation is divided into atomic operations called events. This mechanism should contain improvements to *select()* function based performance. Events are placed in a queue where HIP Daemon could act as a scheduler and pick events sequentially to execution. This could reduce clients from blocking each other during the base exchange. The challenge in this approach is how to prioritize the different events, to prevent blocking.

Other solution could be the use of threading in the implementation. Each client connection could run in its own thread. This solution would prevent blocking well, but with a large number of clients it can be very resource consuming to the server. Also managing the concurrent operation is difficult for the developer. There are available HIP implementations which support threaded operation. In the future it should be studied if such an implementation can be ported to other HIP platform.

From the service variety point of view, using the implementation as a secure WLAN hotspot is the most interesting one. Existing charging schemes could be used in a scenario where HIP server acts as a controller to multiple WLAN bridges. A client connects to a base station which directs all the requests to a HIP capable controller. All negotiation related to the service usage is done with the controller. However, this requires breaking the original idea of the HIP to act only in a point to point connection. The WLAN operation would require changing the IPsec BEET mode to the tunnel mode, so incoming traffic from the clients to the controller could be forwarded securely. This would require also developing a proxy service to the access point, which would keep track of the open tunnels, so the returning traffic could be directed to the right tunnel. Modified HIP firewall could handle this kind of service. [48] [49]

To move the implementation towards actual usage, a few things require some adjustment. The first one is the time based charging. The existing system is pretty coarse and large number of clients may cause problems, because connections just lose the track of common time, which is synchronized after each sent and received chain piece. Some

kind of synchronization mechanism is needed to maintain a reliable service. One solution could be adding synchronization parameters to UPDATE packets when sending hash chain pieces. HIP measures the round trip time of the connection, so the parameter could contain the time at the moment of sending including the measured round trip time. Based on these, both sides could adjust the counters. Other solution could be using some kind of existing time synchronization protocol. This would require binding the moment of sending of the hash chain piece to time instead of duration.

One interesting research subject is mobility. Since HIPL has been ported to Maemo and Android mobile operating systems, it is possible to use this implementation also in mobile devices. During this project there was not enough time and expertise to investigate the compatibility of the implementation in the mobile devices, but theoretically there should be no obstacles to port the implementation to smart phones.

9 CONCLUSIONS

Based on the experiences gained from the implementation, HIP and HIPL seem to provide a suitable development platform for non-repudiable services. HIP has various built-in mechanisms needed to accomplish the non-repudiation which helped a lot during the implementation phase. The protocol specification is pretty loose and the architecture is made quite modular, which helps the development and specialization of the protocol. In the implementation none of the design choices were forced to be changed because of the restrictions of the HIP protocol specification.

On the technical side, the biggest problem of the implementation was the single threaded operation of HIPL. In spite of this restriction it was possible to implement a prototype which fulfilled the functional and non-functional requirements at least in a satisfactory manner. For many requirements, the functionality was achieved but the operation is pretty coarse. However, as a proof of concept type of solution, the implementation fulfilled its purpose. During the development, no major blocking problems were faced, which tells about the good modularity of the HIPL platform.

Results of the performance measurements support the feasibility of the implementation in actual use. If the aforementioned single threaded problem is not taken into account, none of the measurements showed the signs of severe performance problems. No major unexpected behavior of the implementation was found during the testing. Reason for the slow socket operation of the HIPL was not found. If this had been known beforehand, the use of the HIP internal communication mechanisms would have been minimized.

With a few modifications, from the technical point of view the NoRSU implementation is feasible for the operator to use in actual business. However, this requires the operator to define procedures which are still open on the commercial and the legal side. The implementation does not define how the operator should take care of the identity management, since now the identities are based on devices. Also the adequacy of the gathered evidence remains open. There are no exact definitions in the Finnish law about the non-repudiation evidence and their validity in the court. So procedures in the misbehavior situations require more development to transform them into marketing aspects. In the present state, the implementation offers the tools for the customer to get more secure service, but lacks the ability to provide the fully secure and reliable service.

On the whole, HIP offers a good platform to develop non-repudiable service provision for operators. HIP is still under heavy development and the implementations are not yet completely mature for commercial use, but in a few years the situation may be better. It is not very widespread yet, but the trend of the Internet to containing more and more malicious actors may act as a good catalyst for protocols like HIP to spread.

REFERENCES

[1] P. Louridas, “Some guidelines for non-repudiation protocols”, ACM SIGCOMM Computer Communication Review, Volume 30, Issue 5, October 2000, pp. 29-38.

[2] S. Herda, “Non-repudiation: Constituting evidence and proof in digital cooperation”, Computer Standards & Interfaces, Volume 17, Issue 1, January 1995, pp. 69-79.

[3] S. Kremer, O. Markowitch, J. Zhou, “An intensive survey of fair non-repudiation protocols”, Computer Communications, Volume 25, Issue 17, November 2002, pp. 1606-1621.

[4] N. Asokan, ‘‘Fairness in electronic commerce”, Ph.D. thesis, University of Waterloo, Waterloo, Canada, 1998.

[5] I. Ray, I. Ray, “Fair exchange in E-commerce”, SIGecom Exch. Volume 3, Issue 2, March 2002, pp. 9-17.

[6] B. Schneier, “Applied Cryptography”, Second Edition, John Wiley & Sons, 1996.

[7] L. 7.8.2009/617 Law of strong electronic identification and digital signatures (in Finnish).

[8] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, "Host Identity Protocol", IETF RFC 5201, April 2008.

[9] C. Rigney, S. Willens, A. Rubens, W. Simpson, “Remote Authentication Dial In User Service (RADIUS)”, IETF RFC 2865, June 2000.

[10] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, ”Internet Key Exchange Protocol Version 2”, IETF RFC 5996, September 2010.

[11] R. L. Rivest, A. Shamir, L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, Communications ACM, Volume 21, Issue 2, February 1978, pp. 120-126.

[12] P. Nikander, J. Laganier, F. Dupont, “An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)”, IETF RFC 4843, April 2007.

[13] US Federal Information Processing Standard, “Digital Signature Standard”, FIPS 186-3, June 2009.

[14] E. Rescorla “Diffie-Hellman Key Agreement Method “, IETF RFC 2631, June 1999.

[15] S. Kent, R. Atkinson, “IP Encapsulating Security Payload (ESP)”, IETF RFC 2406, November 1998.

[16] S. Kent, R. Atkinson, “Security Architecture for the Internet Protocol”, IETF RFC 2001, November 1998.

[17] P. Nikander, J. Melen “A Bound End-to-End Tunnel (BEET) mode for ESP”, IETF Draft draft-nikander-esp-beet-mode-09, Expired February 2009.

[18] InfraHIP, “Host Identity Protocol for Linux”, [WWW] available in http://infrahip.hiit.fi/index.php?index=about (accessed 10/2011).

[19] B. Adoba et al, “Criteria for Evaluating AAA Protocols for Network Access”, IETF RFC 2989, November 2000.

[20] D. Carrel, L. Grant,”The TACACS+ Protocol Version 1.78”, IETF Draft draft-grant-tacacs-02, Expired June 1998.

[21] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, “Diameter Base Protocol”, IETF RFC 3588, September 2003.

[22] S. Winter, M. McCauley, S. Venaas, K. Wierenga, “TLS encryption for RADIUS”, IETF Draft draft-ietf-radext-radsec-08, Expired September 2011.

[23] S. Keski-Kasari, K. Huhtanen, J. Harju, “Applying Radius-based Public Access Roaming in the Finnish University Network (FUNET)”, In proceedings of TERENA Networking Conference and CARNet User's conference, May 2003. pp. 1-2.

[24] 3GPP TS 23.228 “IP Multimedia Subsystem (IMS) Stage 2”, 3GPP Specification, March 2011.

[25] H. Ventura, “Diameter - Next Generation’s AAA Protocol”, M. Sc. Thesis, Linköping University, Linköping, April 2002.

[26] M. Stanke, M. Sikic, “Comparsion of the RADIUS and Diameter Protocols”, In proceedings of the 30th International Conference on Information Technology Interfaces, June 2008, pp. 893-898.

[27] Cisco, “AAA Servers”, [WWW] available in http://ciscosecurity.org.ua/index.html?page=1587051672/ch11lev1sec2.html (accessed 03/2011).

[28] US-CERT, “MD5 vulnerable to collision attacks”, [WWW] available in http://www.kb.cert.org/vuls/id/836068 (accessed 03/2011).

[29] L. Lamport, “Password authentication with insecure communication”, Communications ACM, Volume 24, Issue 11, November 1981. pp. 770-772.

[30] R. L. Rivest, A. Shamir, “PayWord and MicroMint: Two Simple Micropayment Schemes”, In Proceedings of the International Workshop on Security Protocols, May 1997. pp. 69-87.

[31] T. Heer, “LHIP Lightweight Authentication Extension for HIP”, IETF Draft draft-heer-hip-lhip-00, Expired August 5, 2007.

[32] A. Chefranov, “One-Time Password Authentication with Infinite Hash Chains,” Book of Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics, 2008, pp. 283-286

[33] K. Bicakci, N. Baykal, “Infinite Length Hash Chains and Their Applications”, In Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '02), 2002, pp. 57-61.

[34] I-C. Lin, M-S. Hwang, C-C. Chang, “The General Pay-Word: A Micropayment Scheme Based on n-dimension One-way Hash Chain”, Des. Codes Cryptography, Volume 36, Issue 1, July 2005, pp. 53-67.

[35] D. Cooper et al, “Internet X.509 Public Key Infrastructure Certificate” IETF RFC 5280, May 2008.

[36] C. Ellison, “SPKI Requirements”, IETF RFC 2692, September 1999.

[37] M. Blaze, J. Ioannidis, A. D. Keromytis, “Experience with the keynote trust management system: applications and future directions”, In Proceedings of the 1st international conference on Trust management (iTrust'03), May 2003, pp. 284-300.

[38] M. Blaze, J. Ioannidis, A. D. Keromytis, “Trust Management for IPsec”, In proceedings of Network and Distributed System Security Symposium (NDSS), February 2001, pp. 139-151.

[39] M. Blaze, J. Ioannidis, A. D. Keromytis, ”Offline Micropayments without Trusted Hardware”, In proceedings of the 5h International Conference on Financial Cryptography, January 2001, pp. 21-40.

[40] O. Ponomarev, A. Gurtov, “Stress Testing of Host Identity Protocol (HIP) Implementations”, In proceedings of Third International Conference on Internet Technologies and Applications, September 2009.

[41] S. Heikkinen, S. Siltala, “Service Usage Accounting”, IEEE Vehicular Technology Magazine, Volume 6, Issue 1, March 2011, pp. 60-67.

[42] W. Arbaugh, A. D. Keromytis, D. Farber, J. Smith , "Automated Recovery in a Secure Bootstrap Process", In Proceedings of Network and Distributed System Security, March 1998, pp. 155-167.

[43] S. Deering, R. Hinder, "Internet Protocol, Version 6 (IPv6) Specification", IETF RFC 2460, December 1998.

[44] T. Heer, S. Varjonen, “Host Identity Protocol Certificates”, IETF Draft draft-ietf-hip-cert-12, Expired September 2011.

[45] E. Vehmersalo, “Host Identity Protocol Enabled Firewall: A Prototype Implementation and Analysis”, Master's thesis, September 2005.

[46] F.P, Brooks, P. Frederick, "No Silver Bullet: Essence and Accidents of Software Engineering", Computer, Volume 20, Issue 4, April 1987, pp. 10-19.

[47] G. Banga, J. C. Mogul, P. Druschel, “A scalable and explicit event delivery mechanism for UNIX”. In Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '99), June 1999, pp. 253-266.

[48] T. Heer, S. Li, K. Wehrle, “PISA: P2P Wi-Fi Internet Sharing Architecture” In proceedings of Seventh IEEE International Conference on Peer-to-Peer Computing, September 2007, pp. 251-252.

[49] W. Hu, “Proxy for Host Identity Protocol” M. Sc. Thesis, Aalto University, April 2010.

APPENDIX A: EXAMPLE NORSU CONFIGURATION

```
[ hip_spki ]

issuerhit = 2001:0016:cf10:25f8:9f8a:e55f:b3f8:5b94 // Our Hit
days = 10 // Validity for certificates
duration = 10000 // Duration of NoRSU Service
chainlen = 10 // Length of hashchains
amount = 8 //
type = time // Type of service, time or traffic
unit = s // Unit for service, s or kb
token = 1 // Amount of chains sent per update

[ hip_norsu_param ]
encoding = 3 // Encoding type
name = NoRSUStreamingService // Service name
subtype = 12 // Service type

[ hip_radius ]
ttphit = 2001:0019:765d:4422:189b:8de4:72a6:5633 // Hit of TTP
Server
```