Validating Medusa

a study on the performance and security of a revolutionary approach for survivable trust management

By: Wiebe K. Wiechers

Supervisors: Prof. Dr. W.G. Vree
Ir. S. Daskapan
Ir T.W. Tewoldeberhan
ABSTRACT
This thesis presents a new approach for enabling highly survivable secure communications within multi agent systems and combines two independent research projects aimed at validating this revolutionary approach. The traditional approach for providing secure communications, however efficient, presents a central point of failure and thus reduced survivability. The Medusa approach covered here, which is the focus of the two research projects, aims to capture the advantages of the traditional approach whilst dealing with its deficiencies. The two research projects covered here are the following:

The first project entitled: Simulating the establishment of trust infrastructures in multi-agent systems, presents and simulates a new approach for creating trusted infrastructures within multi agent systems. A bootstrapping protocol initializes a disordered space by turning it into an organized, redundant hierarchical structure, headed by elected security distribution centers with a pool of successors in case of a failure. A simulation was created of the establishment of this hierarchy to judge both the resulting structure and the process of creation in a varying environment. Networks are tested of different scale, type and topology, with different numbers of malicious agents, intent on disrupting the bootstrapping process. The results show the bootstrapping protocols ability to handle the abovementioned constraints. Based on the results some improvements are suggested to improve the bootstrapping protocol further.

In the second project entitled: Ascertaining the security of a distributed survivable trust management protocol, the security of a new approach for enabling survivable secure communications in multi agent systems is validated. The security validation of this approach centers around three security properties: confidentiality, integrity and authentication. Requirements for these security properties are defined for every message generated by this security protocol during its life cycle. A logical analysis of these requirements is followed up by a thorough security validation, based on a model-checking CSP/FDR analysis. Both analyses show that with minor modifications the protocol is able to deliver on its security requirements for the three tested security properties. Finally, the protocol is optimized with possible improvements that increase its efficiency whilst maintaining the security requirements.

The conclusions of these two research projects are combined to form an overall conclusion regarding the viability of the Medusa concept as a practical solution for survivable secure communications. The enhanced survivability of the Medusa concept is proven in a simulated environment and trade-offs with regards to security, autonomy and trustworthiness are excluded. Although more research is required before Medusa is implemented in practice, all conclusions from these two projects indicate that the Medusa concept is a more survivable alternative to present approaches for secure communications in open networks.

INDEX
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABSTRACT</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>4</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>4</td>
</tr>
<tr>
<td>2. RESEARCH ORGANIZATION</td>
<td>4</td>
</tr>
<tr>
<td><strong>CHAPTER 1: MEDUSA</strong></td>
<td>6</td>
</tr>
<tr>
<td>1. INTRODUCING MEDUSA</td>
<td>6</td>
</tr>
<tr>
<td>2. THE BOOTSTRAPPING PHASE</td>
<td>6</td>
</tr>
<tr>
<td>3. THE PREPARATION PHASE</td>
<td>7</td>
</tr>
<tr>
<td>4. THE RESURRECTION PHASE</td>
<td>7</td>
</tr>
<tr>
<td>5. WORKS RELATED TO MEDUSA</td>
<td>7</td>
</tr>
<tr>
<td><strong>CHAPTER 2: SIMULATING THE ESTABLISHMENT OF TRUST INFRASTRUCTURES IN MULTI-AGENT SYSTEMS</strong></td>
<td>8</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>8</td>
</tr>
<tr>
<td>2. SIMULATION MODEL</td>
<td>8</td>
</tr>
<tr>
<td>2.1 CONCEPTUALIZATION</td>
<td>8</td>
</tr>
<tr>
<td>2.2 SPECIFICATION</td>
<td>9</td>
</tr>
<tr>
<td>2.3 TREATMENT, VERIFICATION &amp; VALIDATION</td>
<td>9</td>
</tr>
<tr>
<td>3. EXPERIMENT DESIGN</td>
<td>10</td>
</tr>
<tr>
<td>3.1 BOOTSTRAPPING REQUIREMENTS</td>
<td>10</td>
</tr>
<tr>
<td>3.2 TEST CASES</td>
<td>10</td>
</tr>
<tr>
<td>3.3 PERFORMANCE INDICATORS</td>
<td>11</td>
</tr>
<tr>
<td>4. RESULTS</td>
<td>12</td>
</tr>
<tr>
<td>4.1 NETWORK TOPOLOGY</td>
<td>12</td>
</tr>
<tr>
<td>4.2 NETWORK SIZE</td>
<td>12</td>
</tr>
<tr>
<td>4.3 TRUSTWORTHINESS OF AGENTS</td>
<td>12</td>
</tr>
<tr>
<td>4.4 TRUST CERTIFICATE RELIABILITY</td>
<td>13</td>
</tr>
<tr>
<td>4.5 PRELIMINARY NETWORK TYPE TESTS</td>
<td>13</td>
</tr>
<tr>
<td><strong>CHAPTER 3: ASCERTAINING THE SECURITY OF A DISTRIBUTED SURVIVABLE TRUST MANAGEMENT PROTOCOL</strong></td>
<td>15</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>15</td>
</tr>
<tr>
<td>1.1 MOTIVATION</td>
<td>15</td>
</tr>
<tr>
<td>1.2 RESEARCH ORGANIZATION</td>
<td>15</td>
</tr>
<tr>
<td>2. MEDUSA COMMUNICATION</td>
<td>15</td>
</tr>
<tr>
<td>2.1 COMMUNICATION OVERVIEW</td>
<td>15</td>
</tr>
<tr>
<td>2.2 SECURITY REQUIREMENTS</td>
<td>16</td>
</tr>
<tr>
<td>2.3 COMMUNICATION ANALYSIS</td>
<td>17</td>
</tr>
<tr>
<td>3. CSP/FDR ANALYSIS</td>
<td>19</td>
</tr>
<tr>
<td>3.1 THE CSP/FDR APPROACH</td>
<td>19</td>
</tr>
<tr>
<td>3.2 PROTOCOL VALIDATION</td>
<td>19</td>
</tr>
<tr>
<td>3.3 PROTOCOL IMPROVEMENT</td>
<td>21</td>
</tr>
<tr>
<td>4. CONCLUSIONS</td>
<td>22</td>
</tr>
<tr>
<td><strong>CHAPTER 4: CONCLUSIONS &amp; RECOMMENDATIONS</strong></td>
<td>24</td>
</tr>
<tr>
<td>1. CONCLUSIONS</td>
<td>24</td>
</tr>
<tr>
<td>2. RECOMMENDATIONS</td>
<td>24</td>
</tr>
<tr>
<td>3. FUTURE WORK</td>
<td>24</td>
</tr>
<tr>
<td><strong>APPENDIX A: BOOTSTRAPPING VISUALIZATION</strong></td>
<td>25</td>
</tr>
<tr>
<td><strong>APPENDIX B: SIMULATION MODEL CLASS DESCRIPTION</strong></td>
<td>27</td>
</tr>
<tr>
<td><strong>APPENDIX C: SIMULATION VALIDATION</strong></td>
<td>30</td>
</tr>
<tr>
<td><strong>APPENDIX D: SCALABILITY PERFORMANCE GRAPHS</strong></td>
<td>35</td>
</tr>
<tr>
<td><strong>APPENDIX E: CASPER CODE</strong></td>
<td>36</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>44</td>
</tr>
</tbody>
</table>
Introduction

1. MOTIVATION

Looking at society today most would agree that the industrial age has ended and the information age is upon us. One of the forces driving this transition has been the communications revolution, with its major change agent, the Internet. The rapid development of the Internet in the last decade has influenced businesses, governments and the average day lives of ordinary people. This development is still ongoing; as a matter of fact it is becoming more rapid every day, with the Internet is still growing at frantic pace. Although the myth of Internet traffic doubling every three months, which was assumed at the height of the Internet bubble, has since been disproved [20], a doubling of traffic has been seen with each passing year since 1997 [21]. Future growth rates are uncertain but even conservative estimates (by RHK, Inc. [36]) still display a growth rate of 50%-60% per year for the foreseeable future.

Only a small portion of this massive amount of traffic is secure communication but the need for and number of secure transactions over the Internet is growing fast. This growth is spurred by various circumstances. First of all, every online purchase requires several secure transactions for order information, credit card data, shipping codes, etc. The Internet bubble may have burst but E-commerce is still booming. A study of online merchants over the past twelve months shows a quarterly increase in transactions of close to 10% [29]. Recent estimates by Forrester Research show that yearly online sales will grow from $144 billion in 2004 to $316 billion in 2010 [34].

Second, and perhaps the most important driving force behind the growing number and severity of secure transactions is the growing reliance of traditional critical infrastructures on the Internet such as: finance, power, transportation and off course communication. Although the notion of a digital Pearl Harbor, an attack on critical infrastructures through the Internet, is more hype than substance, it is undeniable that our increasing dependence on the Internet brings forth new requirements with regards to security and availability. Currently the secure communications on the Internet are mostly provided through a centralized Security Distribution Centre (SDC) [28]. An SDC acquires secret keys from its subscribers and facilitates secure communication between them by granting secure session keys when needed. Although such a centralized approach is a very efficient solution it has one important disadvantage: it forms a central point of failure. Without the Centralized Distribution Centre no secure communication is possible, therefore any failure or attack that brings down the SDC is catastrophic. Backup systems can counter this threat to some extent but dedicated backup is an inefficient solution (the spare capacity is not utilized 99 percent of the time) and is ineffective against sustained attacks (because the backup server will be knocked out as well).

Unfortunately the threat to central points of failure has increased over the past five years with the advent of Distributed Denial of Service attacks (DDoS). A "denial-of-service" attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service [5]. By using multiple entities in this attack, a Distributed Denial of Service Attack, the attack becomes very hard to defend against. The devastating power of DDoS attacks was best shown in February of 2000 when in a period of two days, the websites of Amazon, Ebay, Yahoo, CNN, Zdnet, Buy.com and others were brought down for several hours, at a total cost of $ 1.2 billion based on estimates by the Yankee Group [37]. These DDoS attacks are increasing in frequency, severity and sophistication because of DDoS tools with a graphic user interface that allow ordinary users with little or no prior hacking experience to initiate these attacks.

Recapping the present situation, there is an exponential growth in the need for secure communications, but the main approach for providing this service is becoming less and less effective because its main weakness (central point of failure) is more at risk due to an increasing frequency of DDoS attacks [16]. It is obvious that a change is needed.

In this thesis, a universal solution is presented in Medusa: Medusa facilitates survivable secure communications and aims to capture the efficiency advantages of a centralized solution, without a single point of failure. Furthermore it is designed to work in an open environment such as the Internet and, best of all, at little or no extra cost.

Before a concept such as Medusa can be implemented in practice it must be tested vigorously. No business or consumer will leave its sensitive information to a system if they are not 100 percent convinced it can protect their information. This master’s thesis combines two independent research projects both conducted to bring Medusa closer to actual implementation.

The first is a simulation of the creation of Medusa’s trust infrastructure it requires to provide the enhanced survivability. Although earlier research has shown that Medusa can provide the enhanced survivability once the trust infrastructure is in place, the autonomous creation of this infrastructure remains untested.

The second paper is a security validation of Medusa’s internal communication. Even if Medusa can deliver on its promise of enhanced survivability at no extra cost it is useless if a hacker can obtain sensitive information during the Medusa process. Various security properties are tested in this research project.

2. RESEARCH ORGANIZATION

This masters thesis is part of an overall research project intent on proving the viability of Medusa as a practical solution for providing secure communications. As such the overall research objective covers a broad spectrum. The overall research objective is: to validate and increase the ability of the Medusa framework to provide enhanced survivability, without a trade-off loss.

The overall research question is directly based on this research objective:

Does the Medusa framework provide enhanced availability of SDC’s compared to traditional implementations, without a trade-off loss?

A number of sub questions have been derived from this overall question:

1. Does Medusa provide enhanced survivability once its trust infrastructure is in place?
2. Can Medusa autonomously create its required trust infrastructure?

3. Are there any trade-off losses?

The first research question has already been answered in an earlier research project. Medusa can provide enhanced survivability with its trust infrastructure in place [1]. The two independent research projects that are combined in this paper focus on the two remaining questions (questions 2 and 3).

The second research question is the focus of the first research project covered in this paper: simulating the establishment of trust infrastructures in multi agent systems. Before enhanced survivability can be provided Medusa needs to autonomously create its required trust infrastructure.

The second research project covers part of question three: Ascertaining the security of a distributed survivable security protocol. Any security leaks in the Medusa framework would be a serious trade-off loss rendering Medusa useless as a security solution in practice. Therefore the second research projects aims to prove that there is no trade-off loss for security.

The two independent research projects described in this paper are completely self-contained. Both contribute to the same overall research objective but that is where the coherency ends. Both are covered in self-containing chapters featuring separate research objectives, methodologies, results, and conclusions.

Chapter 1 starts of by giving an introduction into Medusa and its workings. Chapter 2 features the first research project, simulating the establishment of trust infrastructures in multi agent systems. The second research project: ascertaining the security of a distributed survivable security protocol; is covered in chapter 3. In chapter 2 and 3, conclusions and recommendations are presented for the independent research projects. These conclusions are combined to form overall conclusions and recommendations in chapter 4.
1. INTRODUCING MEDUSA

As modern networks grow and intertwine the number of critical transactions, which channel through them, increase also in both number and severity. Because of their critical nature these transactions require the establishment of a trust relation between the participating agents. Therefore these critical transactions cannot exist without an efficient trust management mechanism in place. Traditionally, the trust management service in most networks is provided by a centralized authority: the Security Distribution Center (SDC). This centralized approach [28] brings advantages such as efficiency and trustworthiness (an agent only has to trust the centralized entity), which are lost in decentralized [3] and distributed [4] [31] [32] solutions. The downside of having one or multiple centralized SDC’s is that without it the network entities have no means of efficient secure communication. Hence these SDC’s form multiple single points of failure and are at risk from a variety of threats, ranging from “normal errors” to service blocking attacks by nefarious entities. The traditional approach to dealing with a single point of failure is adding redundancy through static replication of the security centers. This approach brings high investments in the dedicated hardware and bad performance, since most of the time the backup hardware is not used efficiently. The Medusa protocol set was developed to capture the advantages of a centralized approach while dealing with its deficiencies [11]. Medusa counters random failures in the SDC, possibly caused by a service blocking attack, through volatile trust management. According to the volatile trust management principle trust is seen as a moveable software object that can be exchanged between several SDC’s [9] [10]. In Medusa this moveable software object, called the trust token, will be moved to another location when the SDC goes down, so the service can continue to be provided. This aspect of Medusa increases the survivability of the SDC. Survivability denotes that a system can continue providing essential services in the presence of attacks or failures, and can recover full services in a timely matter [13]. The Medusa framework creates a pool of possible successors to the current SDC. When a failure occurs at the SDC one of the successors can continue to issue the trust services that are needed by other entities on the network (clients). Normal operation and ensuring the survival of the ad-hoc SDC, through continuous replication and possible migration of its trust token [8], is performed by Medusa’s survivable security management protocol (SMP). Medusa’s bootstrapping protocol is the self-organizing process that leads to the creation of the SDC and its pool of successors. Through interactions between agents (not dedicated to Medusa) a network of unrelated and untrusted agents is converted into a trusted hierarchical structure.

2. THE BOOTSTRAPPING PHASE

During the bootstrapping phase the structure required for the subsequent phases is created, its success or failure is determined by this resulting structure. The bootstrapping phase can be divided into three subphases: Primary Elections, Pool formation, Ratification. A graphical overview of the bootstrapping phase is available in appendix A.

2.1 Primary Elections

Before bootstrapping can begin, a network must exist with a number of agents. These agents are not dedicated Medusa agents, but have background Medusa functionality. Furthermore these agents must have initial but limited access to one or more Central Exogenous Authority’s (CEA). It can provide them with a trust certificate that contains a quantitative indication of their trustworthiness. In the intended Medusa structure one or several chosen agents will perform the centralized trust management service. In the beginning there is no structure and there are no SDC’s. Triggered by self-interest, based on the desire to run reliable, secure transactions in a survivable form, some agents will claim this centralized function and put themselves forth as candidate SDC’s. Since Medusa runs in the background of a primary process the agent must have sufficient capacity available to even consider itself for candidacy. Once an agent opts to become a candidate it informs all other agents of this fact. When all candidates are known the actual voting process takes place. All agents create a list of ballots containing a voting score for each candidate. These scores are based on the absolute trust value given by the CEA, and the agent’s individual level of trust in the CEA itself. If only one CEA is present in the network its trust certificate will be absolute (the candidates with the highest trust value will thus be voted prospect SDC), when more CEA’s are available this is not the case due to agents having different levels of trust in the various CEA’s. After receiving all the votes the candidates individually calculate the score of each candidate by weighting the votes of every agent with its respective trust value and summing up the results. This means that every candidate will have a greater appreciation for the voting scores of the agents it trusts most. The top scoring candidates are the prospect SDC’s.

2.2 Pool Formation

The goal of this sub phase is creating a pool of possible successors to the SDC in case it should at some point cease to exist. To that end the leader invites other agents to join its pool. These agents may be former opponents (unelected candidates) or agents that did not have an active role up to this point. Non-candidate agents are invited to become pool member based on the prospects individual level of trust in them. Former candidates have the benefit of being graded by all agents during primary elections. High scoring but unelected candidates could prove excellent pool members. Nevertheless the prospects deciding factor in inviting a former candidate remains its individual level of trust in these candidates. The invited pool members evaluate whether to indeed become a pool member of the inviting prospect. They must have sufficient capacity available and sufficient trust in the prospect to do so. Once enough pool members agree the pool formation is finished.

2.3 Ratification

The final step in establishing a Medusa structure is ratification of the prospects by end members. All prospects that have managed to form a pool invite agents to subscribe to them by advertising themselves and their pool structure. This pool structure indicates
their possible survivability rate as their pool members are considered future successors. The agents individually decide whether or not to become a client of a certain SDC, or leader. Even when clients have subscribed they may still switch to another leader, which presents an increase in trustworthiness or survivability. When all leaders have invited clients they evaluate their clientele. If it is large enough to warrant the leader’s existence the leader can start providing its security service, if not the leader will disband its pool and group of clients, leaving it and its former pool members available to join new pools or subscribe to other SDC’s.

3. THE PREPARATION PHASE
Once the trust infrastructure has been created it must be maintained. The goal of the preparation phase is to prepare the leader to resist future failures. To this end the leader creates a survival kit, containing a list of its clients, an ordered list of possible successors and the trust token, the combination of the individual secrets of each client. The trust token is divided into several pieces through twisted combinatorial sharing [8]. Each pool member receives the client list, the successor list, and a piece of the trust token. The pieces of the trust token are encrypted and distributed amongst the pool members with non-matching keys in such a way that a majority of the pool members can reconstruct it in case the leader ceases to exist.

4. THE RESURRECTION PHASE
The resurrection phase allows the pool members to resurrect the trust token and appoint the successor as the new leader in case the leader failure. First each pool member determines the successor based on the successor list. Before sending its piece of the trust token to the successor every pool member checks whether the successor is alive. If the successor is operational all pool members send their piece of the trust token to the successor. The successor gathers all pieces and reconstructs the token. With the trust token the successor assumes the leadership role and starts to refresh both the pool member and the client keys. Subsequently the clients acknowledge the new leader by refreshing their secret. The new leader combines the fresh secrets into a new trust token and enters the preparation phase.

5. WORKS RELATED TO MEDUSA
Medusa aims to provide trust relations in a disordered and anarchic space. Other works have a similar aim [3] [4] [31] [32], but do not establish a trust infrastructure. Group Communication Systems (GCS) [2] [23] do offer a trusted infrastructure and some [22], like Medusa, are specifically designed to withstand failures and malicious agents, however these GCS’s feature an unranked infrastructure. A Medusa-like hierarchical structure, baring resemblance to a centralized approach [28] is proposed in Enclaves [15], but it does not feature redundancy or autonomous creation of the infrastructure. Medusa’s redundant hierarchical structure is the result of an election scheme, a topic of ongoing research [6] [7]. It differentiates Medusa from other decentralized and distributed approaches and allows Medusa to operate with efficiency and trustworthiness, without a single point of failure.
Chapter 2: Simulating the Establishment of Trust Infrastructures in Multi-Agent Systems

1. INTRODUCTION

1.1 Motivation
In this chapter a simulation is created of the bootstrapping phase, discussed in chapter 1. Earlier research has shown that Medusa can provide enhanced survivability once its trust infrastructure is in place. Whether Medusa’s autonomous self-organizing bootstrapping phase can create this structure will be tested here.

1.2 Research Organization
Medusa’s promise of enhanced survivability relies on the outcome of the bootstrapping phase. The autonomous and self-organizing nature of this process makes it hard to make predictions regarding the eventual structure without further in depth research, especially since Medusa was developed to operate in a variety of environments. How changes in the environment in which the bootstrapping protocol operates affect the outcome is an important unanswered question.

In this theory testing research the concept of Medusa bootstrapping is tested. The objective of the research is: to validate and improve the Medusa bootstrapping phase by performing a detailed study of its workings and judging its outcome in various environments on predefined structural requirements.

The research method that was chosen for this project is simulation. Testing the bootstrapping phase can be done in a variety of ways, but simulation offers great insight in the protocols inner workings and the possibility of adapting the environment of operation, two specific research objectives.

The simulation model that was designed to simulate the bootstrapping process is covered in section 2. The experimental design is discussed in section 3 while results of the experiments are found in section 4. Finally these results are discussed further in the fifth and final section.

2. SIMULATION MODEL
This section shows the various steps in creating the simulation model. The steps covered here are: conceptualization, specification, treatment, verification & validation.

2.1 Conceptualization
The goal of the conceptualization phase is to create a conceptual model to serve as a basis for the actual simulation model. Before such a conceptual model can be formed the boundary of the system needs to be defined.

2.1.1 System Boundary
To delineate the boundary of the system under scrutiny a number of restrictions were placed on the model and its environment.

The model will only simulate Medusa’s bootstrapping phase, the subsequent preparation and resurrection phases will not be present in the model.

The bootstrapping model will be run on a computer simulated network environment. This network contains some simplifications compared to real life to networks. First and foremost there are no other traffic sources present on the network. All traffic is generated by the Medusa bootstrapping protocol. Second, all notes and links on the network are of similar size and bandwidth.

2.1.2 Conceptual Model
The conceptual object model shown below was created based on the pseudo code supplied by Medusa's designers. This conceptual model shows all object classes that are present in the simulation model and their respective attributes and operations.

```
Class CEA
attributes:
- IDlist
operations:
- send IDlist

Class CE
attributes:
- IDlist
- free capacity
operations:
- request IDlist
- send ballotlist
- become candidate
- become pool member
- become client

Class Candidate
attributes:
- list of ballotlists
operations:
- send candidacy declaration
- process voting scores
- become leader
- become CE

Class Leader
attributes:
- my pool
- my clients
operations:
- invite pool members
- invite clients
- evaluate
- become CE

Class Pool Member
attributes:
- my leaders
operations:
- remove leader
- become CE

Class Client
attributes:
- my leader
operations:
- remove leader
- become CE
```

Figure 1: Conceptual object model

The flow of the model is as follows: the model starts off with all CE’s requesting an IDlist from the CEA. The CEA sends an IDlist to the requester and the CE’s individually decide whether or not to become candidate. All candidates send candidacy declarations to notify the CE’s of their candidacy. Subsequently the CE’s determine their voting scores for each candidate and send them to the candidates contained in a ballot list. The candidates gather old ballot lists in their list of ballot lists and determine the high scoring candidates. These candidates become leader and invite pool members, while the other candidates change back to CE. The new pool members store the ID of their leader in my leaders and the leader stores the ID’s of its pool members in my pool.
Afterwards the leader starts inviting clients, storing their ID’s in my clients. When the leader is finished inviting clients, he evaluates whether to remain a leader. If the leader has not gathered sufficient pool members and clients he will change back to CE and so will his clients and pool members, unless a pool member has multiple leaders.

2.2 Specification
In the specification phase the simulation model was built. Before starting the actual model construction, an effort was made to reduce the models complexity.

2.2.1 Model Reduction
Although much effort was put into creating put into making a very accurate representation of the bootstrapping phase, some key assumptions were made during this process for varying reasons.

The encrypting and decrypting of messages sent between CE’s is not simulated nor is the time that encrypting and decrypting takes taken into account. The reason for not simulating the actual encryption is that it would have proven to great a strain on the system performing the simulation to handle the encryption and decryption of every message sent in the network. The time that the encryption process would take is also not simulated because it has no affect on the eventual structure of Medusa after the bootstrapping phase. Although time itself can affect the eventual structure the amount of time needed for encryption/decryption is negligible compared to other temporal factors in the network (such as different starting times and distance between CE’s).

The different trust values used in the model are random and based on a uniform distribution. When operational both the model and Medusa itself will be run on different forms and types of networks, where any CE can have any trust value. Although in the real world the distribution of trust would possibly be more accurately modeled by a normal distribution it is believed that a uniform distribution of trust is a greater challenge to Medusa’s bootstrapping protocol (since there are no obvious leaders). Hence a uniform distribution is used to test the Medusa bootstrapping phase, as it should prove itself in a “worst case scenario”.

2.2.2 Model construction
The simulation environment used for translating the conceptual model into the simulation model is NS-2 [35]. NS-2 is a discrete event driven and object oriented network simulator, written in C++ and TCL, developed at UC Berkeley. Due to Medusa’s unique approach to providing enhanced survivability a new set of libraries was created for NS-2. These new libraries were written in C++ and represent the different entities that are involved in Medusa’s bootstrapping phase. In NS-2 it is possible to create a network structure consisting of a number of nodes with links between them. Agents can be attached to these nodes to add functionality (for instance a TCP-traffic generator). By creating agents that contain Medusa functionality it is possible to simulate the bootstrapping process on different type of networks.

There are really only two different types of entities in Medusa. One is the so-called Central Exogenous Authority (CEA). This is the entity that provides the initial trust information that is required for bootstrapping. The second type of entity is the computing entity or CE. CE’s are the main entity during the Medusa lifecycle. All CE’s together form the Medusa structure. A CE can take on different roles dependent on its place in the structure. These roles are: Leader, Poolmember, Client, Candidate or None (no role). By making CE and CEA subclasses of Agent, an existing class in NS-2, it is possible to use CE and CEA in NS-2. As stated before NS-2 uses both C++ and TCL. The coupling between the two is both complicated and obscure. For the Class CEA the implementation was kept straightforward putting all of its functionality into a single class. This was done because this entity is only used for starting up the bootstrapping process and does not play a role afterwards. The concept used for the class CE is commonly known as a state pattern [14], which allows the CE to change its state during model operation. In this case all the methods required for the coupling with NS-2 were put into the class while all true Medusa functionality is kept in the class CESTate and its subclasses and is written in clear C++ code. Functionality, which is identical to all roles of a CE, is kept in CESTate, whilst role specific functionality is kept in the relevant subclass. The main advantage of this approach is that if the simulation is expanded in the future intricate knowledge of NS-2 is not required. C++ programming knowledge will suffice. The setup used for this simulation is shown in the conceptual class diagram below:

A thorough description of the different classes of the model is found in appendix B.

2.3 Treatment, Verification & Validation
Now that a simulation model has been created it needs to be determined: how the tests are going to be run, whether the translation of the conceptual model to the simulation model was correct, and if the results from the simulation model are in compliance with reality.

2.3.1 Treatment
Before testing can begin the start up time and run length of the model and the number of replications need to be determined.

Start-up time: because the bootstrapping model can be seen as a finite system determining the start up time is straightforward, there is no start up time.

Run length: determining the run length is also helped by the fact that this is a finite system. The run length will be such for each experiment that the bootstrapping phase has finished and hopefully a structure has been created.

Number of replications: for every experiment of the model will be run four times, and the average of these four observations will be the results of the experiment. The number of four was based on the time required for simulation and the similarity of results with this number of runs.

![Conceptual class diagram](image-url)
2.3.2 Verification
The first step in verifying a model is analyzing the various input variables and verifying whether they occur in reality. For this particular model this step is not very relevant. The main input values are the size and topology of the network, one of the factors that will be experimented with later, and the various trust values. The reason for choosing uniform distributions for these trust values has already been clarified. Networks of various size and topology will be used for testing but all are possible in reality.

The second step in verifying the model constituted analysis of the model logic. This was done by adding print statements to each method in the model code. By analyzing the text printed by the simulation it was determined that the flow model logic was indeed correct. Also the commented simulation code was analyzed thoroughly to remove any flaws in the model logic

Finally the output variables of the model were verified by analyzing them together with Medusa designers. Even they cannot be certain of the outcome of a Medusa bootstrapping process but they can make an educated guess of the structural outcome, which complied with the output variables of the simulation.

2.3.3 Validation
Two validation techniques were used: expert validation and structural validation [26]. The expert validation of the model was conducted by showing the model to experts on Medusa. These experts studied the visualization of the model extensively during multiple simulation runs. Also individual entities were traced throughout simulation runs to check whether their individual behavior was accurate. Structural validation focuses on trivial relations between the models input and internal variables, and the performance indicators that will be measured (the performance indicators that are used in this model will be specified in section 3.3). These relations are tested by changing one of the variables and observing the effect on the relevant performance indicator. A list was generated for this particular model containing the necessary performance indicators and the variables influencing them. This list was discussed with experts on Medusa to determine Medusa’s behavior when a variable is changed. Although it was not possible even for the experts to quantify the change on a performance indicator they could define whether the relation between the variable and the performance indicator was a positive or a negative one (i.e. if the variable increases the performance indicator will increase/decrease). Furthermore it was possible to predict Medusa’s behavior in extreme cases, a variable being zero or 100 percent. The relations and extreme cases were tested in the simulation model and compared with the expected results. Any differences between the two were analyzed to see if the discrepancy could be explained. This process was concluded successfully so that the simulations results are considered reasonably valid for the tested performance indicators. The validation is described in more detail in appendix C.

3. Experimental Design
Now that the simulation model has been discussed and proven valid the focus shifts to the actual experiments. Before setting up the test cases and performance indicators it must be determined what constitutes a successful bootstrapping phase. These bootstrapping requirements are dealt with first, followed by a description of the various test cases. Finally, the performance indicators used for interpreting the outcome of the test cases are discussed.

3.1 Bootstrapping Requirements
The objective of the bootstrapping phase is creating the structure required for the subsequent phases. In this structure a leader is necessary as it provides the centralized service. During the SMP-phase the leader continuously distributes its trust token amongst its pool members in such a way that no individual pool member has access to the secret information and a majority of pool members can reconstruct the information upon the death of the leader. The minimal number of pool members that satisfies these conditions is two. Furthermore a number of clients need to be subscribed to the leader since otherwise there would be no need for the leader and its pool. Two clients would theoretically be enough because they could subscribe to the leader for their secure transactions. However having a leader and two pool members for secure transactions between just two agents would be very inefficient. They might as well use asymmetric keys for their communication and bypass the leader altogether. To justify the existence of the leader and its pool the number of clients needs to be at least one greater than the sum of the leader and its pool members. There is one last essential requirement for the bootstrapping phase although it does not have any direct influence on the eventual structure that is created. As leaders in the bootstrapping process are democratically chosen by a voting majority it is essential that the voters have a choice. Hence there must be more than one candidate during the bootstrapping phase for it to be successful. In short the essential requirements for the bootstrapping phase to be successful are:

- at least one leader
- at least two pool members
- more clients than the sum of the leader and its pool
- more than one candidate

3.2 Test Cases
The objective of these test cases is to test the bootstrapping phase in different environments. Ideally Medusa will be able to operate in any environment, meaning: any type or form of network with any number of agents. The environment may contain malignant agents trying to disturb the voting process. On top of that the security of the CEA’s may differ in various environments. With this in mind a set of test cases was designed. For the initial tests a network will be used with full connectivity and only known agents, resembled closest by a LAN. However a WAN topology will also be tested and preliminary network type tests will be run to see whether the bootstrapping protocol can deal with the extra constraints that may be posed by other predominant network types. All other environmental characteristics mentioned earlier will be tested.

Network Topology: Testing must show that the bootstrapping phase can be completed successfully on any LAN network. Therefore the four major types of LAN topologies and all will be tested: Ring, Bus, Star and Tree. With future network type tests in mind a WAN topology is also tested. This topology consists of nodes linked to supernodes, which are interconnected through a backbone link. It is expected that the Medusa bootstrapping process will function regardless of network topology (as long as sufficient connectivity exists).
Network Size: Medusa must be able to handle networks of different sizes. Tests will be run to verify this and to observe the affects of network size on the success of the bootstrapping process. The network size tests will be run on a WAN topology for on these networks scalability will be of greatest concern. It is expected that the bootstrapping phase can deliver the required structure with different numbers of agents.

To test the scalability performance of the protocol, the strain it puts on the network is analyzed for an increasing number of agents. Earlier communication complexity research has shown that the greatest network load, O(n²), occurs during primary elections, when all agents send a list of ballots to every candidate. This theoretical, network wide, assumption will be tested for a practical network connection: the average backbone link.

Trustworthiness of Agents: Medusa was designed to be able to isolate untrusted agents and candidates during bootstrapping that try to undermine the voting process. In this situation a group of malicious agents puts forth a candidate (or more than one) and try to get it elected by all giving it a maximum score during voting. It is assumed that given this situation an untrusted agent may become a leader and even gain pool members, but only untrusted agents will subscribe to it. This is because clients take the trustworthiness of the leader and his pool into account while deciding whether or not to subscribe to it. This assumption will be tested for increasing numbers of untrusted entities.

Trust Certificate Reliability: the Medusa bootstrapping depends heavily on the integrity of the initial trust values. When an agent can break into a CEA to hack its trust certificate (and increase its trust value) the results may be disastrous. The expectation is that if this were to happen an imposter could become a leader and gather trusted clients.

Preliminary Network Type Tests: A LAN is a good proving ground for bootstrapping but for Medusa to truly be successful bootstrapping should be possible on other types of networks such as: mobile ad-hoc networks and the internet. In a LAN connectivity is close to guaranteed and information on every agent is available. These other networks offer two more constraints that the bootstrapping process must handle: incomplete trust information and reduced connectivity (through lost packets). Medusa was designed to run on any type of network it is therefore expected to handle these constraints to some extent.

3.3 Performance Indicators

When testing the bootstrapping phase in different circumstances performance indicators are required to evaluate the grade of success. The outcome of Medusa’s bootstrapping phase can differ in three ways: the survivability and integrity of the resulting structure, and the “fairness” of the process itself. It needs to be determined which quantifiable factors influence these indicators so that a scheme for grading a bootstrapping process on these performance indicators can be developed. Except for the trustworthiness, trust certificate cases, and scalability performance tests, the following indicators are used:

Survivability: for truly delivering on its promise of enhanced survivability Medusa needs a more developed structure for the subsequent phases. First of all a more rugged structure requires more pools, which means more leaders, and bigger pools, which means more pool members. Second it is important for enhanced survivability that the various pools are connected. This means that some members of a pool, whether it is the leader or pool members, need to be members of another pool and so on. The more pools are interrelated, the more survivable the eventual structure will be. A measurable factor for this property is the number of pool connections, a pool member being represented in more than one pool. Redundant connections are counted only once, as more redundant connections do not add to the level of survivability.

Integrity: When clients subscribe to a leader they confirm that they consider the leader and its pool sufficiently trusted. Therefore as more agents subscribe to the available leader(s) the trust in the structure and thus its integrity is greater. However in a larger network naturally more clients will subscribe to the leaders. Therefore, the deciding factor of a structure’s integrity is the percentage of the total number of agents (that are not yet either pool member or leader) that have subscribed.

Process: As stated while discussing the bootstrapping phase’s essential requirements, agents need the ability to choose. Building on this requirement, it is important that the number of candidates while voting is greater than the number of eventual leaders. This constraint is similar to “survival of the fittest” and means that of all the candidates only the best have become leaders, so the integrity of the leaders will be greater. The factors that will be used to determine the grades for all performance indicators are:

- number of agents (agents)
- number of leaders (ldrs)
- number of pool members per leader (plmbr)
- number of clients per leader (clnts)
- number of candidates (cand)
- number of pool connections (p.c.)

Both the number of pool members per leader and the number of clients per leaders are averages when multiple leaders are present.

Survivability grade (Surv): ldrs * (plmbr+1) * (p.c.+1)

The survivability grade is based on the size and number of all pools and their level of interrelation. For survivability a grade of 6 is achieved when the outcome of the formula is 3, derived from the essential requirements: one leader, two pool members. A grade increase of 1 occurs with a doubling of the formula’s outcome.

Integrity grade (Int): (clnts * ldrs) / (agents – (ldrs * (plmbrs+1))

The integrity grade is consists of the percentage of agents in the network without function that has become client. The minimum number of clients to ratify every leader is the pool size plus one (at least on more client). Therefore the grade 6 is the number of leaders times the pool size plus one again divided by the total nr of agents minus all leaders and pool members. A score of 10 is awarded when the value is 100%, the in between scale is linear

Process grade (Proc): ldrs / cand

The process grade consists of the leader/candidate ratio. A value of 6 is obtained when 25 percent of the candidates become leader (this value was chosen because the top 25% of the candidate list becomes a prospect). A perfect score of 10 is given when the value is five percent while the score scales linearly.

Different indicators are used for the scalability performance, trustworthiness and trust certificate tests. For the scalability performance tests the number of messages passing through the
backbone, connecting the supernodes, determines the load on the network. The following indicators are measured for the average backbone link:

- total nr of message sent (messages)
- average load (messages/time unit)
- peak load duration (time units)

To determine the average load and peak load duration, an unspecified time unit is used. Peak load duration is measured because maximum network load is a network characteristic; the duration of the maximum load depends on the performance of the bootstrapping protocol.

For the trustworthiness and trust certificate tests the emphasis is on determining whether an untrusted leader is elected and manages to gather trusted clients. To judge this the following indicators will be shown (if applicable):

- number of trusted leaders
- number of untrusted leaders
- number of (un)trusted clients per trusted leader
- number of (un)trusted clients per untrusted leader

4. RESULTS

In this section the results of the various experiments will be presented. All tests were run multiple times and the values given in the following tables are the averages of these tests.

4.1 Network Topology

All of the tests for the different topologies feature 30 agents. Table 1 shows the average numbers for each network topology and their scores on the performance indicators.

<table>
<thead>
<tr>
<th>Topology</th>
<th>ldrs</th>
<th>plmbr</th>
<th>clnts</th>
<th>cand</th>
<th>Surv</th>
<th>Int</th>
<th>Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>star</td>
<td>1</td>
<td>1.5</td>
<td>1.25</td>
<td>1</td>
<td>1.25</td>
<td>1.25</td>
<td>1</td>
</tr>
<tr>
<td>bus</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>ring</td>
<td>1.25</td>
<td>3.13</td>
<td>7.88</td>
<td>7.25</td>
<td>6</td>
<td>7</td>
<td>7.5</td>
</tr>
<tr>
<td>tree</td>
<td>1.25</td>
<td>3.5</td>
<td>8.75</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>wan</td>
<td>1</td>
<td>3.75</td>
<td>11.75</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The results in table 1 show that the Medusa bootstrapping protocol does indeed run successfully regardless of topology. It must be noted however that the bus and ring structure at first did not come up with the required structure. Because of the greater maximum distance between nodes in these topologies some of Medusa’s internal time-out timers needed to be relaxed.

4.2 Scalability Size

Scalability was tested by running numerous tests and gradually increasing the number of agents.

<table>
<thead>
<tr>
<th>Scalability</th>
<th>ldrs</th>
<th>plmbr</th>
<th>clnts</th>
<th>cand</th>
<th>p.c.</th>
<th>Surv</th>
<th>Int</th>
<th>Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10.25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>1.5</td>
<td>4.13</td>
<td>12.13</td>
<td>10.25</td>
<td>0.25</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>60</td>
<td>2.75</td>
<td>4.34</td>
<td>11.29</td>
<td>15</td>
<td>2.5</td>
<td>7.5</td>
<td>8.5</td>
<td>7.5</td>
</tr>
</tbody>
</table>

From table 2 a number of conclusions can be drawn. The results show that with 20 or less agents the bootstrapping phase is unsuccessful. The simulation showed that prospects were elected and pool members obtained but the pools were not ratified by enough clients. With more agents the bootstrapping phase does deliver the required structure every time, proving the protocols scalability. As a matter of fact the results show that both the survivability and the integrity of the created structure actually increase when the number of agents grows, the “fairness” of the process does not seem to be influenced by the number of agents.

4.3 Trustworthiness of Agents

In this test a number of untrusted agents were inserted into the network. At least one becomes a candidate while all other untrusted agents award it the maximum score during voting. To disturb the voting process all other (trusted) candidates receive a score of zero from the untrusted agents. Early tests showed that for the voting process to be disturbed by malignant entities two factors are relevant: the number of malignant entities and their average trust values. The malignant entities cannot be completely untrusted if they desire to influence the outcome of the voting process. The tests show that for lowly trusted agents to have any influence on the voting process a vast majority of agents would need to be malignant, not a likely situation. Therefore for further testing the average trust for malignant agents was set at 40% which is assumed to be the maximum trust value malignant agents would be able to acquire from a CEA. The tests were run in a network of 60 agents with an ever increasing number of malignant entities.

<table>
<thead>
<tr>
<th>Table 3. Scalability performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr of messages (messages)</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

Interpreting the results from table 3 it can be concluded that the total nr of messages sent and the average load feature a close to linear increase, indicating a complexity of O(n). The results for 20 agents do not adhere to this conclusion but in this case the bootstrap was unsuccessful (see table 2). The discrepancy with the expected O(n^2) can be explained by the increase in the number of backbone connections that occurs with a sufficient increase in the number of agents. Peak load duration shows polynomial increases which serves as evidence to support the O(n^2) complexity, assumed for worst case network load. A graphical overview of these results is provided in Appendix D.
Table 4. Trustworthiness of Agents results

<table>
<thead>
<tr>
<th>Nr of untrusted agents</th>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>untrusted ldrs</td>
<td>0</td>
<td>1,25</td>
<td>2</td>
</tr>
<tr>
<td>trusted ldrs</td>
<td>2,75</td>
<td>1,5</td>
<td>0</td>
</tr>
<tr>
<td>trusted clnts with trusted ldr</td>
<td>25</td>
<td>15,5</td>
<td>0</td>
</tr>
<tr>
<td>untrusted clnts with trusted ldr</td>
<td>5,5</td>
<td>1,5</td>
<td>0</td>
</tr>
<tr>
<td>trusted clnts with untrusted ldr</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>untrusted clnts with untrusted ldr</td>
<td>5,25</td>
<td>15,75</td>
<td>26,25</td>
</tr>
</tbody>
</table>

Table 4 shows that 10 malignant agents (approximately 15%) cannot disturb the outcome of the voting process. When this number is increased to 20 (1/3 of the total nr of agents) some untrusted leaders are chosen but trusted agents do not subscribe to them, instead most of them subscribe to the trusted leader. Even some untrusted agents subscribe to the trusted leader; because of its high trust value and strong pool. When half the agents in the network are malignant the voting process is completely controlled by the malignant entities, no trusted leaders are chosen. However in this situation the trusted entities choose not to subscribe to a leader, rather than subscribing to an untrusted one. In conclusion, the Medusa bootstrapping protocol can deal with malignant entities, as long as they are recognizable by their trust certificate, what happens if they are not, is tested in the following paragraph.

4.4 Trust Certificate Reliability

This test shows what happens if an agent manages to hack its trust certificate to give itself the maximum allowed trust value, resulting in a non-traceable untrusted leader. Again the tests were run with a network size of 30 agents.

Table 5. Trust Certificate Reliability results

<table>
<thead>
<tr>
<th>various runs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>untrusted ldrs</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>trusted ldrs</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>clnts with trusted ldr</td>
<td>39</td>
<td>21</td>
<td>26</td>
<td>26</td>
<td>17</td>
</tr>
<tr>
<td>clnts with untrusted ldr</td>
<td>0</td>
<td>19</td>
<td>13</td>
<td>7</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5 shows that if but one agent manages to hack its trust certificate the results are worse than in the previous paragraph, as the untrusted leader does manage to gather a group of trusted clients. However the tests also point out that even hacking a trust certificate does not guarantee a malignant agent a leadership role. Also, in each of the runs the majority of clients did not subscribe to the untrusted leader.

4.5 Preliminary Network Type Tests

The following tests were run to see whether the bootstrapping protocol would be able to cope with the extra constraints that are present in other network types: reduced connectivity and incomplete information.

Reduced connectivity does not have an impact on the bootstrapping phase when a message is sent expecting a reply since the sender can simply resend the message if no reply is received. However, when a message is simply broadcast and a reply is not guaranteed the sender will not detect a lost packet, no retransmission will take place, and the message will be lost. There are two of these broadcasts in the bootstrapping phase: when the candidates notify all agents of their candidacy and when the leaders invite clients to become agents. The reduced connectivity is simulated by using a scalable chance variable of not receiving the broadcasted messages for every agent.

Table 6. Reduced Connectivity results

<table>
<thead>
<tr>
<th>% lost</th>
<th>ldrs</th>
<th>plmbr</th>
<th>clnts</th>
<th>cand</th>
<th>p.c.</th>
<th>Surv</th>
<th>Int</th>
<th>Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>2.5</td>
<td>3.6</td>
<td>13</td>
<td>10</td>
<td>1</td>
<td>7.5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>10%</td>
<td>1.75</td>
<td>4.25</td>
<td>19</td>
<td>11</td>
<td>1</td>
<td>7.5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>20%</td>
<td>2</td>
<td>3.67</td>
<td>14</td>
<td>12</td>
<td>1.25</td>
<td>7.5</td>
<td>7.5</td>
<td>6</td>
</tr>
<tr>
<td>30%</td>
<td>2.25</td>
<td>3</td>
<td>12</td>
<td>14</td>
<td>1</td>
<td>7</td>
<td>7.5</td>
<td>6</td>
</tr>
<tr>
<td>40%</td>
<td>1.75</td>
<td>3.25</td>
<td>11</td>
<td>13</td>
<td>1</td>
<td>6.5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>50%</td>
<td>2</td>
<td>2.75</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>6</td>
<td>6.5</td>
<td>6.5</td>
</tr>
<tr>
<td>60%</td>
<td>0.75</td>
<td>2.75</td>
<td>5.25</td>
<td>11</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

The results contained in table 6 shows that the bootstrapping process is quite capable of handling the reduced connectivity. The survivability of the structure is maintained and is only reduced when there is a massive reduction in connectivity. The integrity of the structure is influenced more. This is because the leaders are now chosen without receiving votes from every agent (they might not be the “optimal” leaders) and not all clients receive invitations from every leader. The process indicator does not seem to be influenced by connectivity. The perfect 10 for 60 percent connectivity loss is explained by the very small average number of leaders (some tests yielded no leaders), and unchanged number of candidates, corrupting the results.

The tests for incomplete information were set up as follows: Every agent only has trust information on a number of agents smaller than the total number of agents. Agents will only vote for a candidate when they know the candidate. Prospects will only invite known agents to become poolmember and clients will only subscribe to known leaders.

Table 7. Incomplete Information results

<table>
<thead>
<tr>
<th>kn agents</th>
<th>ldrs</th>
<th>plmbr</th>
<th>clnts</th>
<th>cand</th>
<th>p.c.</th>
<th>Surv</th>
<th>Int</th>
<th>Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>3.5</td>
<td>3.67</td>
<td>14.21</td>
<td>17.75</td>
<td>2.5</td>
<td>8.5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>70</td>
<td>3.5</td>
<td>3.77</td>
<td>12.33</td>
<td>18.5</td>
<td>2</td>
<td>8</td>
<td>7.5</td>
<td>7</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>3.75</td>
<td>15.69</td>
<td>17.75</td>
<td>1</td>
<td>6.5</td>
<td>6.5</td>
<td>9</td>
</tr>
<tr>
<td>50</td>
<td>1.5</td>
<td>1.88</td>
<td>7.38</td>
<td>10</td>
<td>0.75</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

Tables 7 shows a stronger decrease in both survivability and integrity compared to the previous test as the number of unknown agents increases. Obviously incomplete information is a more stringent test for Medusa than reduced connectivity. Nonetheless, the bootstrapping protocol is able to handle incomplete information for a successful trust structure is created as long as more than 50% of the agents in the network are “known”.

13
5. CONCLUSIONS
The objective of this research was twofold: proving the bootstrapping protocol’s ability of establishing a trust infrastructure in varying environments and enhancing its ability to do so. This segregation is also made in this final section. The first paragraph combines all results from testing to prove the concept of Medusa bootstrapping. The second paragraph covers any improvements to the bootstrapping phase inspired by these experiments. Finally the topic of further research is discussed.

5.1 Proving the concept
Medusa is able to handle different network topologies when its internal timers are adapted to take an increase in distance between nodes into account.

The bootstrapping phase is a very scalable process. As long as some minimal number of agents is present, not only will the bootstrapping process provide the necessary structure, the quality of the structure will increase with the number of agents. The strain Medusa puts on a network during bootstrapping is considerable. The average load is linear but the maximum load seems to scale according to O(n^2). This is offset by the fact that the bootstrapping phase is designed to be a one-time process. Once a structure has been created; bootstrapping has served its cause. An improvement to make this scenario possible in practice is suggested in the following paragraph.

Medusa bootstrapping also shows off its ability to cluster any number of untrusted agents that try to manipulate it. Untrusted agents need considerable numbers to have any effect on bootstrapping and even then they cannot coerce trusted agents into an untrusted structure.

The integrity of the trust certificates is Medusa’s “Achilles heel”. When an agent manages to alter its trust certificate the imposter can gain a significant number of trusted clients. In practice the situation of a malignant agent hacking into a certificate authority is very unlikely. Nevertheless the results are not as bad as initially expected. Because of agents’ individual preference, the imposter was not guaranteed a leadership role and never did the untrusted agent gather a majority of the clients. When the network size increases so does the variety of individual preference, meaning that the percentage of clients subscribing to the imposter will become smaller. In the next paragraph suggestions will be made that can further negate this problem in environments were the security of the CEA’s cannot be guaranteed.

Medusa proved very capable of handling reduced connectivity. Incomplete information posed more of a challenge but again the bootstrapping protocol showed the ability to deal with this constraint. Further research for Internet and mobile ad-hoc network types is definitely warranted. Possibly by tuning the bootstrapping protocol further even better results can be obtained.

5.2 Improving the concept
A number of improvements to Medusa have spawned from this research. Two important improvements already came up when building the model and were implemented into the simulation before the tests in this paper were run. Both have to do with an optimal distribution of available trust and capacity resources. In the initial bootstrapping concept “a first come first serve” problem arose when leaders started to invite clients. The first leader to complete its pool and start inviting clients would gather lots of clients (when the leader and its pool are sufficiently trusted) leaving the scraps to the remaining leaders. By allowing clients to switch when an invitation of a better leader comes along the distribution of clients over the available leaders is better and clients end up subscribing to the leader they trust the most. To make sure that agents do not need to go through the hassle of switching leaders when the benefit is minimal, the new leader must be a sufficient increase over its present leader. The other improvement is the notification non-ratified prospects send to the leaders to allow these leaders to possibly fill out their pools with the former prospect and/or its pool members. Before, this step did not take place and valuable resources were wasted (the prospect and its former pool members may well be highly trusted agents).

Other improvements became apparent after the testing phase and are therefore not present in the model. Improvements are required in an environment where the security of CEA’s is not guaranteed. An easy improvement is to require agents to gather trust certificates from multiple CEA’s, which will all be used during elections, pool formation and ratification. In this case an imposter needs to hack numerous CEA’s making its task more difficult. Also an improvement can be made to the prospect selection algorithm. Normally the top 25 percent becomes prospect. A possibility is (especially in larger networks) to take some percentage of candidates that are all sufficiently trusted and randomly select those that become prospects. In this situation an imposter is not guaranteed a prospect role even when it manages to hack its trust certificate (and thus also reducing its incentive to do so in the first place).

Especially during the primary election the bootstrapping process poses a considerable load on the network. Hence it is undesirable to run a bootstrap every time there is a considerable change in the network environment. During its normal operation the Medusa structure should be able to handle an increasing or decreasing number of agents, clients and pool members without having to bootstrap anew. Leaders can choose new pool members from its group of clients and new agents can subscribe to present leaders. At some point however these leaders will become saturated and a new leader should be elected. Since the old leaders were democratically chosen they represent a majority of the network and can therefore justly appoint a new leader. In reality this will likely be a pool member of multiple pools because of its strong relation with a number of leaders. This new leader can then form a pool and be ratified by end members as in any other bootstrap.
Chapter 3: Ascertaining the Security of a Distributed Survivable Trust Management Protocol

1. INTRODUCTION

1.1 Motivation

It is now proven that the Medusa framework can provide enhanced survivability (Chapter 2) [30]. However for Medusa to be successful in practice the Medusa process itself must also be secure. In this context a claim that Medusa is secure, must be asserted based on several precisely defined security properties. The security properties covered in this chapter are: confidentiality, integrity and authentication, all will be specified later.

1.2 Research Organization

Medusa is designed to cover internal security (systems security) and external security (communications security). The internal security is important to Medusa but is also quantified by a CE’s trust certificate. CE’s without adequate internal security will not receive a high trust value from the certificate authority and therefore will not become a leader in the Medusa process. The external security of Medusa is still untested and any lapses of external security may have dire consequences, such as sensitive information falling into the wrong hands. In this paper the focus is solely on the external security.

In this theory testing research the external security of the Medusa process is analysed. The objective is to: assert the security of the Medusa process by performing a thorough security validation on predefined essential security properties and recommend alterations to Medusa to improve either security or efficiency.

To validate the security of the Medusa process, two research methodologies are used. First a general logic based analysis is performed followed by a more thorough model-checking approach by state enumeration (in this process every possible state of the protocol is analysed). The main reason for choosing this combination is its merit in the field of security validation [25].

The three security properties that will be validated in this paper are: confidentiality, authentication or integrity. They are defined here as follows:

Confidentiality: no plaintext data of a message passing between honest entities may be derived by unauthorized entities.

Integrity: any corruption of data contained in a message must always be detected

Authentication: when a message alleges to be from a certain entity, it was indeed originated by that entity.

First an overview is given of the communication during the Medusa lifecycle, its security requirements, and an inspection for obvious lapses with respect to the security properties. Hereafter Medusa is subjected to a CSP/FDR state enumeration analysis to find any possible attacks an intruder could utilize to subvert Medusa’s confidentiality, authentication or integrity. Following, the protocol is analysed to see whether these properties can be asserted more efficiently. Finally, the results of the various analyses are summarized and discussed further.

2. MEDUSA COMMUNICATION

A gross overview of the entire Medusa process has already been covered in chapter 1. In this section the specific methods within Medusa that feature communication will be highlighted and analysed for their security requirements. Subsequently, the Medusa protocol in its current form is analysed to ascertain whether it provides these requirements without an active intruder intent on compromising them.

2.1 Communication overview

Medusa was designed as a single solution for security services. This does not automatically imply that it is also one single security protocol. A security protocol is defined as: a prescribed sequence of interactions between entities to achieve a certain end [25]. For a security validation the Medusa framework can be looked upon as a combination of security protocols occurring either sequentially or concurrently. To reduce complexity the number of agents participating in a protocol is kept to a minimum. For instance during the primary elections phase all CE’s communicate with all candidates. This can be modeled as a protocol between one CE and one candidate, which is run between all CE’s and candidates. The analysis of the latter situation is much less complex but does not reduce the quality of the security validation. This section presents an overview of the different protocols by showing the sequence of messages (the interactions between the participating entities). For coherency with the Medusa pseudo code the methods in which the messages are sent are shown as well.

2.1.1 Bootstrapping Phase

Four protocols are run in a sequential manner during the bootstrapping phase. The final three security protocols match sub phases from the Medusa overview. However the primary elections subphase also contains a protocol between a CE and a CEA, the initialization protocol.

Initialization:
1. CE → CEA : IDlist request
2. CEA → CE : IDlist 

{Initialize}

In this initial protocol every CE acquires an initial IDList from central exogenous authorities containing the ID’s and trust certificates of most CE’s in the network space.

Primary Elections:
1. Cand → CE : candidacy declaration
2. CE → Cand : acknowledge receipt
3. CE → Cand : ballot list
4. Cand → CE : acknowledge receipt

{ModPublishCand ModCastBallots}

The “would be” leader declares its candidacy to every CE in the network space by broadcasting a candidacy declaration over the network. A receiving CE acknowledges receipt of the declaration and calculates the trust value of each candidate leader in its scope.


Subsequently it sends its ballot lists to the candidate, which is acknowledged by the candidate upon reception.

**Pool formation:**
1. Prospect → CE : covenant  \{ ModProcessVotes \}
2. CE → Prospect : acknowledge receipt
3. CE → Prospect : signed covenant  \{ ModVouches \}
4. Prospect → CE : acknowledge receipt

The candidate leaders create a final score list of votes to determine whether their own score is high enough to become a leader and if so invites other CE’s to sign their covenants. Together with the covenant the leader sends a symmetric key for future communication. The prospective pool member acknowledges receipt. If it agrees to become a pool member it signs the leader’s covenant and returns it to the leader, if not it sends a rejection to the leader.

**Ratification**
1. Prospect → CE : subscr. Invitation  \{ ModRatification \}
2. CE → Prospect : acknowledge receipt
3. CE → Prospect : subscr answer  \{ ModSubscs \}
4. Prospect → CE : acknowledge receipt
5. Prospect → CE : symmetric key  \{ ModMembership \}
6. CE → Prospect : acknowledge receipt

The leader invites clients to subscribe by sending them his signature file. Alternatively a client can request an invitation from a leader when it has not received one. The prospective client acknowledges receipt and responds to the leader’s invitation. The leader acknowledges and adds the subscribing client to its client list. It subsequently sends a symmetric key for efficient secure communication back to the client, who acknowledges receipt.

**2.1.2 Preparation**

There are two security protocols that run during the preparation phase: the trust token sharing protocol and the alive protocol. Although the two protocols feature similar entities and run concurrently they are in fact two distinct security protocols. This is because the alive protocol is run much more frequent than the trust token sharing protocol. The first is run only periodically when a significant change to the secret has occurred (due to for instance a Secssr : trust token
5. Secssr → Plmbrs : acknowledge receipt

When a pool member suspects that the leader has died it double-checks this by sending an alive request. If the leader does not respond, it informs the other pool members. Subsequently all pool members that believe the leader to be dead, acknowledge this fact to all pool members. If enough (more than half) pool members acknowledge, the pool members send their piece of the trust token to the first live entry in the successor list. The successor acknowledges upon reception.

**Key & Secret Refreshment**
1. Leader → Clnt/Plmbr : new key  \{ ProcRefreshSecr \}
2. Clnt/Plmbr → Leader : ackn. receipt
3. Clnt/Plmbr → Leader : updated secret
4. Leader → Clnt/Plmbr : ackn. receipt

After the reconstruction of the trust token the successor sends new keys to the clients and pool members. Upon receiving their new keys the clients resend their secret to their leaders.

**2.2 Security Requirements**

As stated in the introduction Medusa will be evaluated for three specific security properties: confidentiality, integrity and authentication. Of these properties integrity should be upheld for every message. If the contents of a message can be tampered with a disruption of the Medusa process is easily achieved. The other two properties, confidentiality and authentication, may or may not be required depending on the type of message. These properties will be evaluated per method, since a method generally contains one message sent and acknowledged. For authentication separate forms are distinguished: one-way authentication, signifying that the initial sender should be authenticated to the receiver, and two-way authentication meaning that the receiver should also be authenticated to the sender.

**Initialize:** Correct authentication both from the CE to the CEA and from the CEA to the CE is essential in the method. The CE needs to be certain that it is indeed obtaining its IDlist from the desired CEA while the CEA must establish the identity of the receiver before it transmits the information to the CE. Confidentiality is an issue, although later in the process most CE’s will share their trust certificates for privacy reasons they should retain the right to keep their certificates private.

**ModPublishCand:** Since the candidacy declaration does not contain any sensitive information confidentiality of the message is not an issue here. Authentication however is a relevant security property here. The candidate needs to be authenticated to the receiver to ensure that it knows who the candidate is.

**ModCastBallots:** It is debatable whether confidentiality truly is an issue in this method; anyone willing to obtain a CE’s ballot list could simply declare itself a candidate. However as with the
method. Initialize for reasons of privacy the confidentiality of the ballot list should be guaranteed so that the ballot lists remain unknown to outsiders of the Medusa process. The voting CE should be correctly authenticated to the receiving candidate, since the candidate was already authenticated to the CE during its declaration of candidacy; two-way authentication is not required within the method.

ModProcessVts: This method only features communication for candidates that go on to become prospects. These prospects send a covenant for signing to a number of prospect pool members. This covenant is clearly meant only for the pool members and should therefore remain confidential. Next authentication should be established from the prospect to the future pool member. The authentication of the pool member to the prospect can be established in its reply, which is sent in ModVouches.

ModVouches: As stated with the previous method the pool member should be authenticated to the prospect. Confidentiality does not seem to be required since only a signed hash of the covenant is returned to the prospect. However another (untrusted) prospect could use such a covenant to fool other CE’s into believing it has a strong pool. Therefore the hashed and signed covenant should remain confidential.

ModRatification: As in the method ModPublishCand the leader broadcasts its subscription request over the network, thus confidentiality is not an issue. Because the clients need to know to whom they will or will not subscribe the leader should be correctly authenticated to the receivers.

ModSubscribe: The leader must correctly establish the identity of its subscriber. If the client does not subscribe the message contains a simple no answer and confidentiality is not an issue. If the client does subscribe it sends a hashed secret to the leader, this should remain confidential as it the proof that any key it receives from this point on came from its leader. In this method a client could also send a subscription request to the leader for such a message there are no requirements with regards to authentication or confidentiality.

ModMembership: The subscription-acknowledge message also contains the symmetric server key that the client can use for efficient communication through the SDC (the leader) and therefore must be kept confidential. Authentication of the leader to the client is essential, as the client must be certain that its key was issued by its leader. The authentication of the client to the leader already occurred in ModSubscribe.

ModPreparation: Although the different parts of the trust token cannot be opened with the keys sent with them confidentiality must still be maintained for else an intruder could acquire the clients’ secrets by intercepting enough pieces of the trust token to reconstruct it. Authentication is also essential and must be established both ways. The leader must be certain that he is indeed sending the trust token to the pool members whilst the pool members must only accept the new token when they are convinced their leader has sent it. If not an intruder can send a fake part of the token to some pool members and disrupt their ability to reconstruct it, destroying Medusa’s redundacy.

ModSelfAssess: The security requirements for “alive” messages are limited. Confidentiality is not required, the message simply shows that the leader is still functioning; it does not contain sensitive information. For active alive requests, authenticity of the requestor to the leader is not necessary; anyone may know that the leader is functioning. Authenticity from the leader to the pool members is required; the pool members must know that it is the leader sending out alive signals and not some imposter.

ModCheckCondition: The requirements for the active alive check were covered in the previous method, there are no. Informing the other pool members of the leader’s demise can also be achieved with few security requirements. Only authenticity from the sender to the receiver is required. The receiving pool members must be certain that the declaration of death was sent by a pool member so that agents outside the pool cannot confuse it with false declarations.

ProcSendToken: The sending of the secret has very strict requirements. Confidentiality is essential since together the pieces of the secret contain highly sensitive information. Also two-way authentication must be established because the pool members must ascertain that they are indeed sending their piece of the trust token to the successor, whilst the successor must know the origin of the pieces it receives.

ProcRefreshKeys: Since this is in essence a key-exchange protocol again both confidentiality and two-way authentication need to be established. Only the receiver and the leader may know the value of the new key, the receiver must be certain that the key came from the leader while the leader must establish that the one it sends the key to is indeed the required recipient.

ProcRefreshSecr: As implied by its name the secret contains sensitive information, requiring confidentiality. Furthermore the secret must be sent to none other than the respective leader, and the leader must be sure of the identity of the sender. This leads to the conclusion that two-way authentication is required.

The following table summarizes the different requirements for each method that were established in the previous text. For Integrity and Confidentiality a “+” value signifies that the respective property is not required, while a “-” signifies the opposite. For authentication “1-way” means that the sender must be authenticated to receiver, and “2-way” represents a two-way authentication requirement.

<table>
<thead>
<tr>
<th>Method</th>
<th>Authent.</th>
<th>Integrity</th>
<th>Confid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModPublishCnd</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ModCastBallots</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModProcessVts</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModVouches</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModRatification</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ModSubscribe</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModMembership</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModPreparation</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModSelfAssess</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ModChkCndition</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcSendToken</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcRefreshKeys</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcRefreshSecr</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

2.3 Communication Analysis

In this section the communication during the Medusa process is specified. The specification shows the keys, nonces, hash
functions and other security tools used in Medusa’s security protocols. Following each specification a thorough analysis is performed to check whether the requirements discussed in the previous section are indeed provided. First a quick overview of the various security tools and their abbreviations is shown:

A, B : identity of a or b
M : the actual message
N : nonce generated by a
PKa, SKa : public and secret (private) key pair of a
Kab : symmetric key shared by A and B
\{m\} SKa : message encrypted with SKa
TS : timestamp
H(M) : a hash of M

2.3.1 Bootstrapping

Initialization:

It was decided not to give a specification of the Initialization protocol. The implementation of this protocol will undoubtedly differ with each Central Exogenous Authority. It is assumed however that a CEA in the Medusa environment has some protocol in place which offers both confidentiality and 2-way authentication. This assumption is valid because the secure distribution of certificates is present on the Internet today.

Primary Elections:

1. Cand(A) → CE(B) : \{M, A, Na \} SKa, \{H(M, A, Na) TS\} SKa
2. CE(B) → Cand(A) : \{H(M, A, Na, Nb) TS\} SKa, Nb
3. CE(B) → Cand(A) : \{M, B, Na\} PKa, \{H(M, B, Na) TS\} PKa
4. Cand(A) → CE(B) : \{H(M, B, Na, Nb) TS\} SKa, Na

For ModPublishCand (m1 & m2) integrity is achieved by creating a hash of all the contents of the message. Any distortions in the message body can therefore be detected. The Candidate is authenticated to the CE by via its secret key. No other CE could have encrypted this message other than the Candidate. The candidate to the leader is required. This is established because message 1 is encrypted with the leader’s private key. ModSubscribe (m3 & m4) requires confidentiality. This is achieved by encrypting the whole message with the clients private key, the hashed secret is encrypted with the prospects public key to provide authenticity. ModMembership provides confidentiality by encrypting the message with the client’s public key. Although no private key is used authenticity is still guaranteed because the leader returns the hashed secret (stored in M) which no one knows other than the leader and its client.

2.3.2 Preparation

Trust Token Sharing:

1. Ldr(A) → Plmbr(B) : \{M, A, Nb\} Kab, \{H(M, A, Na) TS\} Kab
2. Plmbr(B) → Ldr(A) : \{H(M, A, Na, Nb) TS\} Kab, Nb

ModPreparation (m1 & m2) provides both authenticity and confidentiality through the use of a symmetric key between the leader (A) and the respective poolmember (B). Through the acknowledgement of reception by the pool member, two-way authentication is achieved.

Alive:

1. Ldr(A) → Plmbr(B) : \{M, A, Nb\} Kab, \{H(M, A, Na) TS\} Kab
2. Ldr(B) → Plmbr(A) : \{H(M, A, Na, Nb) TS\} Kab, Nb

The alive messages sequence in modSelfAssess (m1 & m2) is very similar to the sequence in ModPreparation even though its security requirements are much lighter. Authenticity is guaranteed by the use of symmetric keys (actually so is confidentiality which is not even required, but since symmetric keys are much more efficient then asymmetric pairs, they are used here as well).

2.3.3 Resurrection

Resurrection:

1. Plmbr (A) → Ldr(B) : \{M, A, Nb\} Kab, \{H(M, A, Na) TS\} Kab
2. Plmbr (A) → Plmbr(C) : \{M, A, Na\} Kac, \{H(M, A, Na) TS\} Kac
3. Plmbr (C) → Plmbr(A) : \{H(M, A, Na, Nb) TS\} Kac, Na
4. Plmbr (A) → Sccssr(C) : \{M, A, Na\} Kac, \{H(M, A, Na) TS\} Kac
5. Sccssr (C) → Plmbr(A) : \{H(M, A, Na, Nb) TS\} Kac, Na

modCheckCondition’s (m1, m2 & m3) first message has no security requirements. The second does require authenticity and confidentiality provided through the symmetric keys (the leader issues keys for the pool members to use in private communications). procSendToken (m4 & m5) also achieves authenticity and confidentiality via its use of symmetric keys. Two way authentication is guaranteed through the acknowledge message (also encrypted with the symmetric key) by pool member (A).

Key & Secret Refreshment:

1. Ldr(A) → CE(B) : \{H(M, A, Na, K1) TS\} SKa, \{H(K1, A, Na) TS\} SKa
2. CE(B) → Ldr(A) : \{H(K1, A, Na, Nb) TS\} Kab, Nb
3. CE(B) → Ldr(A) : \{H(M, B, Na) K2, \{H(M, B, Na) TS\} K2
4. Ldr(A) → CE(B) : \{H(M, B, Na, Nb) TS\} K2, Nb

ProcRefreshKeys (m1 & m2) provides authenticity and confidentiality via the symmetric keys of the former leader which
the successor (A) has successfully rebuilt. Now the new leader uses the old keys to distribute new symmetric keys between itself and CE (B) who is either a client or a pool member. ProcRefreshKeys (m3 &m4) subsequently achieves authenticity and confidentiality through the use of the new symmetric keys, just issued by the new leader.

2.3.4 Provided Requirements
Table 9 shows a whether or not the requirements from table 1 were provided. A plus (+) indicates that the requirement was met; a minus (-) shows it was not, three x’s indicate (xxx) that this specific property was not required for this method.

Table 9. Security Analysis results

<table>
<thead>
<tr>
<th>Method</th>
<th>Authent</th>
<th>Integrity</th>
<th>Confid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModPublishCnd</td>
<td>+</td>
<td>+</td>
<td>xxx</td>
</tr>
<tr>
<td>ModCastBallots</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModProcessVts</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModVouches</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModRatification</td>
<td>+</td>
<td>+</td>
<td>xxx</td>
</tr>
<tr>
<td>ModSubscribe</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModMembership</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModPreparation</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ModSelfAssess</td>
<td>+</td>
<td>+</td>
<td>xxx</td>
</tr>
<tr>
<td>ModChkCndition</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcSendToken</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcRefreshKeys</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ProcRefreshSecr</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

As seen from this table in most cases the requirements were met. It must be noted that this is in an environment where no malignant entity is purposely trying to subvert these properties. Looking at the protocols’ specification many of the security tools may not have been required to establish the above properties. However, when an active intruder is operating, these extra tools may very well be essential to Medusa for continuing to provide authenticity, integrity and confidentiality.

One requirement in the table is not met: the sender-receiver authenticity requirement in modCastBallots. Authenticity was not achieved because the message was only encrypted using the receiver’s public key. If the message is instead encrypted with the sender’s private key, as seen in modProcessVotes, both authenticity and confidentiality will be guaranteed as is required. In further testing this improvement will be adopted.

3. CSP/FDR Analysis
Now that the various security requirements for the Medusa protocol have been established and validated in a “friendly” environment, the analysis continues in an environment where active intruders will attempt to subvert the Medusa process. To catch replay attacks by the intruder, the authentication requirement is enhanced with the following condition: if an entity A believes it has run a protocol once with another entity B then B has run the protocol with A exactly once. To validate Medusa in a hostile environment the CSP/FDR approach is introduced [25]. Following this introduction Medusa’s ability to uphold its security requirements in a hostile environment is analyzed. Finally the CSP/FDR approach is used to determine whether the efficiency of Medusa’s various security protocols can be increased.

3.1 The CSP/FDR Approach
The CSP/FDR approach consists of the CSP (Communicating Sequential Processes) process algebra and the model checker (FDR) (Failures/Divergences Refinement). CSP is a mathematical framework for the description and analysis of systems consisting of components (processes) interacting via the exchange of messages [18] [24] [27]. The fact that CSP is designed specifically for describing parallel processes communicating with each other makes it inherently suited for modelling security protocols. In CSP, Clients, Pool members and Leaders can be modelled as processes performing a sequence of actions. By default the network will deliver a message to its specified destination but an intruder is active on the network. This intruder conforms to the Dolev-Yao model [12]. This model introduces an attacker able to manipulate messages passing through the network by deleting, faking, redirecting replaying and so on, only bound by cryptographic constraints. This intruder is present in each run of the security protocol and tries everything in its arsenal to subvert the security properties of the protocol. With regards to cryptography, perfect encryption is assumed. An intruder can only decrypt a message when it possesses the required key, it cannot guess the key or perform dictionary or brute force attacks.

The model checker FDR is a commercial product developed by Formal Systems (Europe) Ltd [33]. FDR compares two descriptions: a specification of “wanted” behaviour and an implementation to determine whether every trace in the implementation conforms to a trace in the specification. If FDR comes up with a trace that is not in the specification it returns this trace as a counter example. When evaluating security protocols this trace represents a course of action the intruder can take to subvert one of the protocols security requirements.

For generating the CSP language a high level compiler was used: Casper [19]. Casper takes a fairly abstract description of a security protocol (similar to the description used in 2.3) and generates the corresponding CSP description. By using Casper both the time required for generating the CSP description and the likelihood of errors in the description is greatly reduced.

3.2 Protocol Validation
In this section CSP/FDR is used to validate the various security protocols described earlier. Only the confidentiality and authenticity requirements can be checked with this approach. However when a message is correctly authenticated and contains a signed hash of its contents, integrity should also be guaranteed. The protocol specifications of paragraph 2.3 can be run in several systems, containing various initiators and responders. Generally the following scenarios are a reasonably complete list of the checks worth making [25]:

1. An Initiator A, and a responder B
2. An Initiator A, and a responder A
3. An Initiator A, a responder A, and an initiator B
4. An Initiator A, a responder A, and a responder Bob
5. An Initiator A, two responders B
6. Two Initiators A, and a responder B

All of these systems will be tested for each Medusa security protocol. When an attack is found in scenario 1 and a solution is possible it will be implemented before further scenarios are tested.
The figure below shows a description of the Casper code for one of the protocols (Primary Elections):

The definition of the security tools used in the protocol is shown in the Free variables section. The knowledge of the actors participating in the process is defined in Processes. Subsequently the actual protocol is described under Protocol description. The description is quite similar to the description used throughout this chapter. Specification shows the security requirements for this protocol. Secret delineates the values that should remain confidential between which actors. Agreement describes the authentication requirement. Agreement(a, b, []) means that a should be authenticated to b in such a way that when b believes it has run the protocol with a, a has indeed run the protocol with b exactly once. Weaker forms of authentication are also possible, has run the protocol with a, a has indeed run the protocol with b.

3.2.1 Bootstrapping

Primary Elections: Table 10 shows that FDR did not come up with any attacks on the primary elections protocol in any of the scenarios.

Table 11. Pool Formation results

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>MdPrcssVts</td>
<td>MdVchs</td>
</tr>
<tr>
<td>1 Attack !!!</td>
<td>no att fdn</td>
</tr>
<tr>
<td>2 no att fdn</td>
<td>no att fdn</td>
</tr>
<tr>
<td>3 no att fdn</td>
<td>no att fdn</td>
</tr>
<tr>
<td>4 no att fdn</td>
<td>no att fdn</td>
</tr>
<tr>
<td>5 no att fdn</td>
<td>no att fdn</td>
</tr>
<tr>
<td>6 no att fdn</td>
<td>no att fdn</td>
</tr>
</tbody>
</table>

In the first scenario an attack is discovered on the authentication requirement of ModProcessVotes. The attack proceeds as follows:

1. Pros(A) → Int : {M}SKaA,Na,Ka,int{PKint,{H(M,A,Na)TS}PKa} 2. Int(A) → CE(B): {M}SKaA,Na,Ka,int{PKb, {H(M,A,Na)TS}PKb} 3. CE(B) → Int(A): {H(M),SKaA,Na,Ka,int, {H(M,A,Na)TS}Kb,int} 4. Int(A) → CE : {H(M),SKaA,Na,Nb,TS}Kb,int, Na

This attacks presents a rather unlikely situation where prospect A invites the intruder into its pool. The intruder subsequently redirects this invitation to B posing as A. The intruder acts as though the symmetric key it is issued by A is instead a key between A and B. B finishes the protocol with the intruder and believes it is now a pool member of A. This is not the case because in fact it did not run the protocol with A. Although it is unlikely that a prospect would actually invite an intruder, this attack can be negated by adding the message destination in the part of the message encrypted with A’s secret key. So the protocol is changed as follows (the change is in the message):

1. Pros(A) → CE(B): {{M}SKaA,Na,Ka,int}PKb, {H(M,A,Na)TS}PKa
2. CE(B) → Pros(A): {H(M),SKaA,Na,Nb,TS}SKb, Nb
3. CE(B) → Pros(A): {H(M),SKaA,Na,Nb, TS}Kb, Na
4. Pros(A) → CE(B): {H(M),SKaA,Na,Nb,TS}SKb, Nb

After this change was adopted no more attacks were discovered in any of the scenarios.

Table 10. CSP/FDR Analysis results

<table>
<thead>
<tr>
<th>Method</th>
<th>Auth</th>
<th>Int</th>
<th>Con</th>
<th>Attack?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModPublishCnd</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModCastBallots</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModProcessVts</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>Attack !!!</td>
</tr>
<tr>
<td>ModVouchers</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModRatification</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModSubscribe</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModMembership</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModPreparation</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
<td>Attack !!!</td>
</tr>
<tr>
<td>ModSelfAssess</td>
<td>1-way</td>
<td>+</td>
<td>-</td>
<td>no attack found</td>
</tr>
<tr>
<td>ModChkCndition</td>
<td>1-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ProcSendToken</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ProcRefreshKeys</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
<tr>
<td>ProcRefreshSecr</td>
<td>2-way</td>
<td>+</td>
<td>+</td>
<td>no attack found</td>
</tr>
</tbody>
</table>
Ratification: It can be concluded from table 10 that FDR did not find any attacks on the Ratification protocol.

3.2.2 Preparation

Trust Token-Sharing:

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModPreparation</td>
<td>ModPreparation</td>
</tr>
<tr>
<td>1. Ldr(A) → Int(B) : {M, A, N_b} K_{ab}, {H(M, A, N_b), TS} K_{ab}</td>
<td></td>
</tr>
<tr>
<td>2. Plmbr(B) → Int(A) : {H(M, A, N_a), TS} K_{ab}, N_b</td>
<td></td>
</tr>
<tr>
<td>3. Ldr(A) → Plmbr(B) : {M, A, N_b} K_{ab}, {H(M, A, N_b), TS} K_{ab}</td>
<td></td>
</tr>
</tbody>
</table>

In scenario 4 Casper detects the following attack on the authentication of ModPreparation:

1. Ldr(A) → Int(B) : {M, A, N_b} K_{ab}, {H(M, A, N_b), TS} K_{ab}
2. Plmbr(B) → Int(A) : {H(M, A, N_a), TS} K_{ab}, N_b
3. Ldr(A) → Plmbr(B) : {M, A, N_b} K_{ab}, {H(M, A, N_b), TS} K_{ab}

This is a textbook replay attack. The intruder poses as B and A sends the secret. The intruder sends this to B twice and B responds accordingly. Now A thinks it is still running the protocol with B while B thinks it has already run it twice. The data integrity is still in tact because the message is still the same. Whether this attack is possible in practice depends on the implementation of the protocol. If B saves the nonce supplied by B for a long as the timestamp is valid, or saves the last timestamp it has received, this replay attack is impossible.

The attack can also be solved regardless of implementation by adapting the protocol as follows:

1. Ldr(A) → Int(B) : {M, A, N_b} K_{ab}, {H(M, A, N_b), TS} K_{ab}
2. Plmbr(B) → Ldr(A) : {H(M, A, N_a), TS, N_b} K_{ab}.

If the nonce generated by B is sent to a encrypted, and subsequently returned to B. Every time B receives its nonce it can be certain that A has run the protocol.

Although the above attack does not cause any discernable damage (as long as the validity of the time stamp is shorter than the intervals of refreshing the token) to the Medusa process, it is recommended that either of the above solutions is adopted. Both solutions are easily implemented and with either solution strong authentication is guaranteed.

Alive: Judging from its specification this protocol should produce similar results to the former. Although it has weaker security requirements than the previous protocol the requirement shared by the two, authentication, was the requirement that was attacked. However, Casper has a special authentication requirement: Aliveness. This requirement states that if B thinks it has successfully completed a run of the protocol with A, then A has previously been running the protocol (A may have thought it was running the protocol with someone other than B). Aliveness is perfectly suited for the Alive protocol. Using the Aliveness requirement, no attacks are found on the protocol in any scenario.

3.2.3 Resurrection

Resurrection: For the CSP/FDR tests the first line of this protocol was left out. The failed alive check does not give the intruder extra tools for disturbing the rest of the protocol and the requirements for the alive check itself are already covered. By leaving out this check the number of states that must be checked by FDR is greatly reduced. Compared to the specification in 2.3 the protocol is enhanced with an extra pool member. It is believed that the extra pool member may grant the intruder extra capabilities for subverting the security requirements.

Judging from the results in table 10 no security lacks were discovered in the critical Resurrection protocol. It must note however that this does not absolutely guarantee a successful resurrection. If enough pieces of the token have been altered or destroyed the successor is not able to recover the token. However judging from the previous analyses there is reason to assume that this scenario is possible unless the systems security of the pool members is found lacking.

Key & Secret Refreshment: As seen in table 10 no attacks were found in the Key Sharing protocol. Therefore it can be concluded that all communication in the Resurrection phase adhere to the required security properties.

3.3 Protocol Improvements

In this Chapter CSP/FDR is used to check whether the security requirements for the various Medusa sub protocols can still be achieved if these sub protocols are “slimmed down”. For each sub protocol the optimized protocol will be shown together with an analysis of the changes made to the protocol. The main goal for this section is to remove Medusa’s dependency on timestamps especially in the Bootstrapping phase. Although limited time synchronization may be possible, even in open and ad-hoc networks, it would be a great improvement to the protocol if the time synchronization dependency were removed.

3.3.1 Bootstrapping

Primary Elections: The analyses of this protocol showed that the protocol can be improved and still deliver all security requirements. The resulting protocol is as follows:

1. Cand(A) → CE(B) : {M, A, N_a} K_{ab}, {H(M, A, N_a)} K_{ab}
2. CE(B) → Cand(A) : {M, B, N_b, A, N_a} K_{ab}, {H(M, B, N_b, A, N_a)} K_{ab}
3. Cand(A) → CE(B) : N_b

Comparing this protocol to the previous a number of improvements are visible. First and foremost the timestamps have been removed without sacrificing required security. Any replay attacks are avoided through the use of nonces. The protocol has also become much smaller, requiring only three messages with the final message being very small and unencrypted. To achieve this the second message is encrypted with a public key while its hash is signed with a private one, guaranteeing both authenticity and confidentiality. Finally both the identity of the candidate and that of the sender are added to the second message to avoid intricate “man in the middle” attacks on authenticity.

Pool Formation: the pool formation protocol also showed some room for improvement:

1. Pros(A) → CE(B) : {M, A, K_{ab}} K_{ab}, {H(M, A, K_{ab})} K_{ab}
2. CE(B) → Pros(A) : {H(M)} K_{ab}, B, N_b K_{ab}, {H(M, B, N_b)} K_{ab}
3. Pros(A) → CE(B) : N_b
The improvements to this protocol are quite similar to the ones found in the primary elections protocol. An important difference is the lack of a nonce in the first message (which therefore is also not returned in the second message). This can be done because in this protocol the symmetric key serves is sent by and returned to the successor. Instead of sending an encrypted hash of the previous message, only an unencrypted nonce is returned.

**Key & Secret Refreshment:** The final security protocol is the key/secret sharing protocol. Considering its similarity to the token sharing protocol it may well have some room for improvements.

1. Ldr(A) → Plmbr(B) : {H(M), A, Na} Kab, {H(Kab,A)} Kab
2. CE(B) → Ldr(A) : {M, Na} Kab, {H(M, Na)} Kab

As was expected the protocol has been improved considerably. It now contains only three messages with the last message being an unencrypted nonce. Also, there are no more timestamps in this protocol. Even though this protocol may not run as often as others in the Medusa life cycle, any improvements in its efficiency should still be implemented.

### 4. Conclusions

The analyses in section 2 have shown that the Medusa protocol can provide the security requirements: confidentiality, integrity and authentication, in an environment without an active intruder. Further analysis in section 3.2 shows that with some minor alterations, Medusa can provide these requirements even when an intruder is out to subvert the Medusa process. Finally in section 3.3 further alterations were suggested to the various sub protocols to increase their efficiency whilst maintaining the security requirements.

These alterations have lead to a protocol that is not dependent on time synchronizations during the bootstrapping phase. In this phase, time synchronization poses a real problem, and the fact that Medusa is able to do without it is a great plus. Furthermore if for some reason time synchronization in the preparation and resurrection phases is unavailable, the Medusa protocol can be adapted to avoid time synchronization altogether.

It must be noted that this security validation only covers the security properties: confidentiality, integrity and authentication. If other security properties are desired, they too should be validated in a similar manner. This is important to bear in mind especially with regards to the suggested improvements to the various protocols for reasons of efficiency. These “new” protocols are slimmed down to a minimum whilst still providing confidentiality, integrity and authentication. When another security property is desired the improved protocols should be evaluated and possibly new functionality must be added to ensure
that the new security property is satisfied, this will result in an efficient solution.
Chapter 4: Conclusions & Recommendations

1. CONCLUSIONS
The objective of this research is to validate the ability of the Medusa framework to provide enhanced survivability, without a trade-off loss. Two separate research projects were conducted both contributing to the completion of this objective. A number of conclusions were drawn from these projects, which carry answers to two of the research questions linked to this research objective.

The results of the first research project answer the research question: can Medusa autonomously provide its required trust infrastructure? The Medusa bootstrapping phase, that establishes the trust infrastructure, shows the ability to autonomously establish the required structure in varying environments. It can handle networks of different size and topology. Furthermore when malicious agents attempt to disturb the voting process they need significant numbers to do so. Even then only the untrusted agents subscribe to the untrusted leader, leaving the untrusted agents clustered together in the resulting structure. The only way a malicious agent can infect the trust infrastructure is by hacking its trust certificate. In this situation the agent can be voted a leader, and gather trusted clients. However, in practice, this situation is very unlikely because of the integrity of the trust certificates. Finally the Medusa bootstrapping protocol is able to handle extra constraints that it may encounter when operating on open or ad-hoc networks. The bootstrapping phase was successful in dealing with both reduced connectivity and incomplete information.

In the second research project Medusa’s security was put to the test. This project in part answers the research question: are there any trade-off losses. Sacrificing security for survivability would be an ill-advised trade-off. The project concludes that with minor alterations the Medusa protocol is a very secure process that is able to provide the analyzed security properties: confidentiality, integrity and authentication. Medusa provides these requirements in environment with or without an intruder actively trying to subvert its security properties.

Regarding the question of trade-off losses some answers can also be abstracted from the first research project. Its results show that there are no trade-offs for autonomy (the bootstrapping process needs no human intervention) and trustworthiness (under ordinary circumstances no trusted clients subscribed to untrusted leaders).

Overall it can be concluded that when combining this research with earlier survivability research on Medusa’s later phases, the objective of validating the Medusa framework to provide enhanced survivability, without a trade-off loss has been brought closer to completion. The first part of this objective has been completed successfully, albeit in a simulated environment, whereas the trade-offs question still requires some work, which will be discussed in section 3 (future work).

2. RECOMMENDATIONS
Both projects also suggest improvements to Medusa to increase its survivability, security or performance. With the adoption of these recommendations Medusa’s ability to provide enhanced survivability without trade-off losses is increased further. The first research project suggests a number of enhancements to the bootstrapping phase.

For a better distribution of resources, clients should be allowed to switch leaders when an invitation of a better leader comes along. Also, prospects that do not become leader should notify other leaders of its own identity and that of its pool members.

In environments where the security of a CEA cannot be guaranteed Medusa can be improved to avoid an imposter hacking its trust certificate and becoming a leader. One solution in this situation is to require candidates to assemble certificates from multiple CEA’s. Any imposter trying to hack its certificate now needs to hack multiple certificates to subvert the Medusa process.

Because the bootstrapping phase puts considerable strain on the network a Medusa trust infrastructure should be able to adapt to a changing environment to avoid having to bootstrap anew. It is recommended to empower the “council” of leaders, who represent a voting majority of all agents, to adapt the Medusa structure as necessary.

The recommendations that have spawned from the second research project are focused on increasing the performance of the Medusa protocols without sacrificing any of the tested security properties. The result is an improved Medusa protocol that is no longer dependent on time synchronization. Especially in the bootstrapping phase, achieving time synchronization can be problematic. Therefore it is recommended that this requirement be removed, by adapting the Medusa protocol. In the subsequent phases time synchronization is easily achieved and can be utilized to decrease the number of messages required during Medusa’s operation.

3. FUTURE WORK
Possible future work has been derived from both research projects. For the first research project, tests need still be performed to prove the Medusa concept on the Internet and mobile ad hoc networks. Only there will Medusa truly demonstrate its superiority over existing trust management mechanisms. Furthermore research should be conducted into the trust structure’s ability to deal with a changing environment. The aforementioned recommendation regarding this situation is still a mere concept and requires further development.

For the security validation performed in the second research project some future work is also possible. Although the CSP/FDR approach used here is very reliable way to uncover vulnerabilities in a security protocol it is not mathematical proof that, given the assumptions of the model, a security requirement is ensured. This proof can be obtained by using rank functions [17]. If such proof is desired creating it using rank-functions techniques is a topic of future work.

Finally, coming back to the overall research objective some future work is required to ensure the absence of trade-off losses in the Medusa approach. Other possible negative affects that should be excluded through further research are: privacy and cost.
Appendix A: Bootstrapping Visualization

In this appendix the bootstrapping phase is shown graphically by showing images of the simulation visualization.

When a CE decides to become a candidate (after receiving its IDlist) it broadcasts a candidacy declaration (red message) over the network. In the visualization the candidates are recognizable by their label (candidate).

Upon receiving a candidacy declaration a CE calculates its voting score for the respective candidate and adds the score into its ballot list. When all candidates have identified themselves the CE’s send their ballot lists (brown message) to all candidates. When the candidates have gathered all ballot lists, each candidate calculates the voting scores for all candidates. The primary elections phase is now complete.

1. PRIMARY ELECTIONS

The bootstrapping phase starts off with all CE’s sending an IDlist request (black messages) to the CEA. Upon receiving a request the CEA sends an IDlist (blue message) to the requestor.

The top candidates become prospects (red candidates). They start inviting CE’s (purple messages) to join their pool as pool members.
The invited pool members decide whether to actually join a leaders pool. The ones that do are recognizable by their label (pool member), the following number indicates their leader (pool members that join multiple pools have more numbers). The pool members send the leader a reply to the invitation (beige message).

3. RATIFICATION

When a prospect has gathered sufficient clients it broadcasts an invitation to subscribe (yellow message) to all CE’s. The CE’s send a reply to the invitation (black message).

One of the leaders did not gather sufficient clients to warrant its existence. This leader (leader 9) sends a message (black message) to all of its clients and pool members, and all other leaders informing them of his demise. The other leaders possibly invite the former leader or its former pool members as pool members of their own, and/or they invite them and the former clients as clients (yellow message).

This figure represents the structure after the bootstrapping phase. There are two leaders, leader 3 has 5 pool members and leader 15 has three. Their pools are not connected because no pool member is present in both pools. Leader 3 has 10 clients; leader 15 has 7.

This is a visualization of the structure after clients have been invited. Clients can change leaders even after subscribing to a leader when they receive an invitation from a “better” leader.
Appendix B: Simulation Model Class Description

In the following text the different classes of the simulation model will be discussed, covering the respective classes’ most important attributes and methods. In essence this is a specification of the conceptual model shown in chapter 2, only in an object oriented programming language, operations are entitled methods. At the start of each subparagraph an overview of the class’s structure is supplied; only the methods and attributes required for understanding the model will be covered. The text accompanying the overview elaborates further on the structure and shows the flow of the model by explaining the order in which the respective methods are called. Methods and attributes of a class under scrutiny are printed in *italics*. A methods body is not printed when it is mentioned in the text.

1. Class CE

**Attributes:**
- certificate IdList
- int numNodes

**Methods:**
- createIdList()
- sendIdList()
- command()
- recv()

The class CE consists of three methods, the first one that is called is *createIdList()*). This method is called directly from the simulator when simulation starts. An IdList is created that contains a trust certificate for all CE’s on the network. This certificate contains the CE’s id, a random trustvalue (taken from a uniform distribution between 1-100), and a random indication of the CEA that issued the certificate (taken from a uniform distribution between 1-100). When a message for the CEA comes in the method *recv()* is called. All messages that are sent during the bootstrapping process are encapsulated in the structure Message. This structure contains entries for the values that need sending and an indicator showing the type of message being sent. The method *recv()* evaluates the type of message and takes the corresponding action. For the CEA there is only one type of message that requires action to be taken: a request for IdList. When this occurs the method *sendIdList()* is called. The CEA sends a certain percentage of the IdList (default = 100%) and the number of CE’s, *numNode*, in the network to the requestor.

2. Class CEState

**Attributes:**
- CE* CE_
  - int typeOfTimer
  - int[] CEAlist
  - int freeCapacity

**Methods:**
- CETimer timer
- changeState(State s)
- procSend()
- setColor(int clr)
- setLabel(String label)
- command()
- recv()

When a CE’s constructor is called (from the simulator) the CE creates a CEState (None) to which the *state_* attribute points. Furthermore it creates a randomn (between 1-100) *freeCapacity* and a *CEAlist*, a list of random trustvalues (again between 1-100) that represent the trust this CE has in each of the CEA’s.

As stated before CE takes care of the interaction with the simulator. Therefore some of CE’s methods, are not very relevant for the bootstrapping process. *setColor()* and *setLabel()* are strictly for visualization purposes and do exactly as their names suggests. *command()* and *recv()* are also present here and function similarly to the ones in CEA. If *recv()* is called the message is past on to *state_* (CEState).

The two most important methods are *procSend()* and *changeState()*). *procSend()* is called from CEState when a message needs to be sent to other CE’s. *changeState()* is also called from CEState and results in the CE changing its state by having the attribute *state_ point to a different subclass of CEState* (for instance Leader). The *CETimer* attribute is a timer that is required to synchronize the timers in CEState and its subclasses with the simulation environment. When *CETimer* expires the method *timerExpire() in CEState* is called.

3. Class CEA

**Attributes:**
- CE* CE_
  - int typeOfTimer
  - int[] Ballotlist

**Methods:**
- reqIdList()
- StoreIdList(certificate []
- considerCandidacy()
- modCastBallots(certificate)
- procTrustVal(certificate)
- sendBallotList()
- procReceive(Message)

CEState handles the Medusa functionality that is similar for all states. When CEState is created it saves a reference to its creator in the attribute *CE_*. The first method that is called is *reqIdList()*). This method is called directly from the simulator. The respective *CE_* sends an IdList request to the CEA and subsequently receives an IdList. This results in *procReceive()* being called from *CE_* in *procReceive* the received message is evaluated and in this particular case *storeIdList()* is called. The IdList is stored in *CE_* to prevent its contents from being lost when a state change occurs. After storing the IdList *considerCandidacy()* is called. This method the agent considers whether to become a candidate based on a number of factors. If the CE’s freeCapacity is higher than the threshold (default is 30) it will create a random (between 100-100) integer desire. If the result of the addition of desire and its own trustvalue is high enough (default is 115) the CE will change its state to Candidate.

*modCastBallots()* is called when the certificate of a Candidate is received. *procTrustVal() is called from within modCastBallots*. The trustvalue of the candidate is weighted with the CE’s trust in the issuing CEA. This result together with the id of the candidate is combined to form a *TI*, This *TI* is returned to modCastBallots and added to the *Ballotlist, typeOfTimer* is set to 1 and the timer in CE is reset. When the aforementioned timer expires, the method *TimerExpire()* is called from *CE_*. This method evaluates the attribute typeOftimer, which indicates the action to be taken. In
this case the action is sendBallotList(). The BallotList is put into a message and sent to all candidates via procSend().

4. Class Candidate

Attribute:  
- int [] TRw
- int [] votespercandidate
- int [] myPool

Methods:  
- modPublishCand()
- modProcessVotes(TI [], int)
- modRatification(int, int)
- modVouches(certificate, int)
- modSubscribe(certificate [])

When a Candidate is created the method ModPublishCand() is called. A message containing a candidacy declaration is broadcast over the network. The Candidate remains dormant until the first BallotList is received. Following this occurrence modProcessVotes() is called, its body containing the BallotList and the id of the CE that sent it. The Candidate weights the ballots in the Ballotlist with the weighted trustvalue of the voter and puts them into the TRw, which is a list of weighted trustvalues for all candidates. This process is repeated as more ballots are received (all ballots are added to the values in the TRw list). Everytime a candidate receives a vote its entry in the list votespercandidate is increased by one. When all votes are received the values for every candidate in the TRw list are divided by the respective values in the votespercandidate list. This means that even if some candidates did not receive votes from every CE their value is still valid. The resulting TRw list is sorted and if a candidate find itself within some top percentage (default is 25) of the list this candidate is a prospect leader (its state does yet not change).

The prospects loop through the rest of the TRw list to find possible pool members. If a candidates TRw value is within a certain percentage of the top prospects score (default is 66%) and if the prospect trusts him as well, meaning that its weighted trustvalue is over a certain threshold (default is 50), the candidate is sent an invitation message. This message contains the prospects id and a minimal capacity (default is 25) that the CE must have available to become a Poolmember.

It is possible that the maximum number of poolmembers (default is 5) has not been invited after looping through the TRw list. In this case the prospect weights and sorts its IdList (without the values of other prospects and already invited poolmembers). The prospects invites poolmembers of the top of this list until the maximum number of poolmembers has been invited.

The method modRatification() is called when a reply to an invitation is received. The body of this method contains the id of the sender and its answer. When the invitee agrees to join the pool, the id of the new poolmember is added to the myPool array. When answers to all invitations have been received the candidate checks the size of its pool. If it has acquired sufficient poolmembers (default = 2) the prospect changes its state to leader, passing myPool to the new state.

modVouches() and modSubscribe() only play a role when the candidate does not go on as a prospect. ModVouches() is called when a pool invitation message is received. The CE checks its freeCapacity with the capacity required for joining the pool. If the CE has enough freeCapacity and trusts the leader it sends a positive reply and changes its state to Poolmember, if not it sends a negative reply.

modSubscribe() is called when a Candidate receives a client invitation message. The weighted trustvalue of the leader and the average of its poolmembers are combined to form an overall trustvalue (66% leader, 33% pool). If this value is sufficient (default is 50) the Candidate sends a subscription message and changes its state to client.

5. Class Leader

Attributes:  
- int [] myPool

Methods:  
- procSendInvit()
- modMembership(int)
- modRatification(int, int)
- evaluateSelf()  
- removeLeader(certificate [])
- removeClient(int)

A Leader calls procSendInvit() upon its creation. With this method it broadcasts a subscription invitation message over the network and sets a timer upon which evaluateSelf() will be called. When a subscription message is received modMembership() is called, adding the client id to the CL or Client List.

evaluateSelf() is called to judge whether the leader has acquired enough candidates to warrant its existence. If this is not the case (default = at least one greater than its nr of poolmembers + 1) the leader sends a disband message to its pool members and clients, and informs other leaders of its demise and the identity of its former pool members. Finally, it changes its own state to None.

RemoveLeader(certificate []) is called when such a disband message is received by a Leader. If the leader has not yet reached its maximum number of poolmembers it sorts the weighted trustvalues of the deceased Leader and its former poolmembers and invites new poolmembers if they are sufficiently trusted.

RemoveClient(int) is called when a leader receives an unsubscribe message from a Client. The Leader then removes the client from the CL.

6. Class Poolmember

Attributes:  
- int [] leaders

Methods:  
- modVouches(certificate)
- removeLeader(int)

When an instance of Poolmember is created it always has one leader in its array of leaders. The id of this leader is passed through the constructor. When the respective Poolmember receives a request to join another pool modVouches() is called. The CE evaluates whether its freeCapacity and the leaders weighted trust value are high enough. If so it creates a vouching message containing a “yes” answer, adds the new leader to leaders and reduces its freeCapacity. If either of the above conditions is not satisfied it sends a similar message with a negative answer. removeLeader() is called when the Poolmember receives a disband message from one of its leaders. The respective leader is removed from leaders and the Poolmember checks whether it has any leaders left. If not it sends a subscription request to all leaders and changes its state back to None.
7. Class Client

Attributes: certificate myLeader
Methods: modSubscribe(certificate)
           removeLeader(int)

Instances of the class Client are created with a certificate of their respective Leader, stored in myLeader, which is passed through the constructor. modSubscribe() is called when an invitation is received from another leader. The Client compares its present leaders weighted trustvalue with that of the new leader. If the new leaders value forms a sufficient increase (default is 5) in trustworthiness the Client sends an unsubscribe message to its present leader and a subscription message to the new one. If the increase is insufficient or nonexistent the Client does nothing.

removeLeader() is called when a disband message is received. The Client subsequently sends a subscription request message to all leaders and changes its state back to None.

8. Class None

Methods: modVouches(int)
           modSubscribe(certificate)

The class None has no attributes. Naturally it does inherit attributes from its superclass CEState. The two methods it contains are modSubscribe() and modVouches(). Their implementation is identical to the similar methods in the class Candidate. The first method performs the necessary operations for becoming poolmember, the second the operations for becoming a client.
Appendix C: Simulation Validation

During model validation it was determined whether the model is accurate enough for its intended purpose, validating the Medusa bootstrapping phase. The validation process was complicated by two issues specific to this particular simulation model. First of all there was no historical data available on the Medusa bootstrapping protocol. Historical data often plays a crucial role in a validation process. To get around this issue a more subjective form of validation was used by choosing validation techniques that permit validation through expert opinions rather than historical data. Furthermore, as the model will be used for validating the Medusa bootstrapping phase, the line between validating the simulation model and validating Medusa would often blur.

In part because of these issues and in part due to temporal constraints it was decided to only validate the simulation for the performance indicators used to determine whether the bootstrapping is successful. This means that if the simulation is used for other purposes in a later stage of the research (for instance to determine the time that is needed to create the Medusa structure) it needs to be validated again for the relevant performance indicators. Two methods were used for validation: expert validation and structural validation.

1. EXPERT VALIDATION

As mentioned earlier the validation of this simulation model of the Medusa bootstrapping phase was highly dependent on the expert opinions on Medusa. A number of techniques were used together with these experts to prove the validity of the model for the required performance indicators.

The first technique used for the expert validation, was animation. The graphical display of the model’s behavior was analyzed extensively by the experts to judge the validity of the models overall behavior and the flow of messages in particular. The flow of messages is well suited for graphical analysis since the different messages are displayed with distinct colors by the model graphical display.

Second, the experts judged the face validity of the model. For this validation technique the experts moved beyond the graphical representations and also judged the various numerical outputs from the model. Specifically the experts compared the resulting structure to the input parameters and judged whether the structure and the process leading to its creation was as expected.

Finally, the experts focused on the behavior of specific entities in the model. By following the behavior of individual CE’s throughout the models life cycle, they determined whether their specific behavior was accurate.

Generally, the results of the expert validation showed that the model gave an accurate representation of the models behavior. Some internal values for the model were fine tuned during this process. The most important change that spawned from the expert validation was an alteration to the voting process. Initially the candidates when weighting the various ballot lists would use the respective CE’s absolute trust value. To better reflect Medusa behavior, this was changed to having candidates use their relative trust value (the absolute trust value weighted by their individual level of trust in the issuing CEA) for weighting the different ballot lists.

2. STRUCTURAL VALIDATION

Structural validation focuses on trivial relations between performance indicators and the models input and internal variables. These relations are tested by changing one of the variables and observing the effect on the relevant performance indicator. If the effects on the performance indicators are in stroke with reality for every variable in the model the simulation is structurally validated.

For this model a list was generated containing the necessary performance indicators and the variables influencing them. This list was discussed with experts on Medusa to determine Medusa’s behavior when a variable is changed. Although it was not possible even for the experts to quantify the change on a performance indicator they could define whether the relation between the variable and the performance indicator is a positive or a negative one (i.e. if the variable increases the performance indicator will increase/decrease). Furthermore it was possible to predict Medusa’s behavior in extreme cases, a variable being zero or 100 percent. In the following text these relations will be specified per performance indicator and the reasoning of the experts will be made clear. Subsequently a table will show the simulations results on these structural tests, which will be analyzed followed by some concluding remarks on the structural validity of the simulation.

2.1 Number of Candidates

\( \text{number of CE's} \): in Medusa every CE decides for itself whether to become a candidate or not based on its free capacity, its trustworthiness and its “desire” to become a leader (an untrusted CE can become a candidate it so chooses). If there are more CE’s in the network more CE’s will become candidate so the relation is positive. The extreme condition tests are fairly straightforward, no CE’s yields no candidates, an infinite number of CE’s will yield an infinite number of candidates. However, the latter is impossible to test.

\( \text{initial trust values} \): as stated before part of the decision of whether a CE becomes a candidate is based on its trustworthiness determined by its initial trust value. Hence, if the initial trust values increases the number of candidates will increase as well, which is a positive relation. As for the extreme condition tests, a 100 percent trust for every CE does not automatically mean that every CE will become a candidate (some CE’s may not desire candidacy). There will be many candidates, but not 100 percent. Although as stated before lowly trusted CE’s can become candidates if they desire to do so, no trust entirely is to big of a hurdle to overcome, meaning that no trust leads to no candidates.

\( \text{free capacity} \): a CE needs to have a certain amount of free capacity for it to consider candidacy. This means that an increase in free capacity leads to an increase in the number of candidates and thus to a positive relation. However the increase or decrease should not be as great as with the initial trust values since the candidates do not actually take their free capacity into consideration when determining whether or not to become a candidate. The extreme condition should be as follows. No free capacity means that no CE will consider candidacy so there will be no candidates. A 100 percent free capacity means that every CE will consider candidacy but since the actual consideration process itself is not influenced the increase will be fairly small.
**free capacity threshold:** this variable is the minimal amount of free capacity a CE needs to have to consider candidacy. It’s effect on the number of candidates is the exact opposite of free capacity. Hence, its relation to the number of candidates is negative one, no threshold means a few extra candidates and a threshold of 100% means no candidates at all.

**desire to become leader:** for the performance indicator number of candidates this variable acts similarly to the initial trust values. In short this means that there is a positive relation, 100 percent desire yields many candidates, zero percent desire yields none.

**candidacy threshold:** the candidacy threshold is the actual threshold that a CE needs to surpass to become a candidate. The two factors that, when added together, must be higher than the candidacy threshold are: initial trust and desire. The relation between this threshold and the number of candidates is negative. If the threshold is 100% there will be no candidates if there is no threshold there will be many candidates but not 100% because some CE’s may still lack the required capacity to even consider becoming a candidate.

### 2.2 Number of Prospects

Although the number of prospects is not a performance indicator that will be measured it is an important interim indicator for the number of leaders. The intricate relation between number of prospects and number of leaders will be clarified when effects of the number of prospects on the performance indicator number of leaders is discussed in the relevant paragraph. In this section the variables influencing the number of prospects will be discussed.

**number of candidates:** in Medusa a certain percentage of the total number of candidates becomes a prospect. As the number of candidates increases so will the number of prospects, a positive relation. Likewise no candidates will mean no prospects and all candidates will lead to many prospects but always only a certain percentage of the total number.

**candidate-prospect percentage:** this is the actual percentage of candidates that moves on to become a prospect. The relation between this percentage and the number of prospects is also positive. The extreme conditions of zero and 100 percent lead to no and many prospects respectively.

### 2.3 Number of Leaders

The number of leaders is a somewhat complicated performance indicator. Of the total number of candidates a certain percentage will become prospects, which try to acquire pool members. If enough pool members are acquired the prospect becomes leader and invites clients. If the leader gets enough clients it stays leader if not the leader reverts to ordinary CE status while its poolmembers and clients are redistributed amongst the remaining leaders. This means that the performance indicators number of clients and number of pool members per leader do not directly influence the indicator number of leaders since these contain the possibly redistributed clients and poolmembers. However a lot of the variables influencing the number of leaders and the number of clients/pool members are similar.

The number of leaders is influenced by a great number of variables. Also the total number of leaders is always relatively small meaning that an increase of even one leader is a large percentual increase. This leads to the fact that for some of these variables the relation will not show in testing because the number of leaders stays the same. The relations for this performance indicator should therefore be interpreted as greater or equal and vice versa (i.e when the initial trust values are greater the number of leaders becomes greater or stays equal).

**number of prospects:** the previous description shows that the number of prospects is a cap for the number of leaders but it does not mean that variables that increase the number of prospects will automatically increase the number of leaders since the prospects still need to gather sufficient pool members and clients to become and remain a leader. It is even possible that the amount of leaders decreases when the number of prospects rises because with more leaders, and as a result a greater total number of pool members there will be less clients available to ratify all these leaders. Hence the relation between number of prospects and number of leaders is unclear, and will not be tested. The extreme conditions however are clear; leading to no leaders for both no prospects and all prospects, since in the latter situation there are no pool members or clients available.

**minimal nr of poolmembers:** this variable constitutes the amount of poolmembers a prospect needs to become a leader. The relationship with the number of leaders is negative as more poolmembers required may result in a smaller number of leaders. For this variable the extreme condition of 100 percent minimal nr of poolmembers means that this variable is equal to the max number of poolmembers per leader. This would lead to a small number of leaders or none at all. Zero poolmembers required would lead to a number of leaders close or equal to the number of prospects.

**max number of poolmembers:** a prospect leader invites a number of poolmembers equal to its maximum allowed. If a prospect is allowed to invite more poolmembers the chance of it acquiring the minimal number of poolmembers required increases: a positive relation. Zero max poolmembers will lead to no leaders as the minimum nr of pool members will not be reached. 100 percent max poolmembers will also lead to little or no leaders because when all CE’s are invited to become poolmember and most agree, too few CE’s will be left to become clients.

**free capacity:** this variable acts exactly opposite to the previous. The relation is positive no free capacity yields no leaders, 100 percent free capacity yields a possible increase in the number of leaders.

**initial trust values:** the initial trust values of a leader are taken into account by clients in their decision whether or not to subscribe to the leader in question. A higher initial trust increases the likelihood clients will subscribe to leaders thus possibly increasing the number of leaders; a positive relation. The extreme condition of no trust leads to no leaders, 100 percent trust might show an increase in the number of leaders.

**trust in CEA’s:** every CE has a different level of trust for the certificate authorities that provide the initial trust value. This variables relation to the number of leaders is similar to the
positive relation described above, its behavior in extreme conditions however is not. 0 percent trust still leads to no leaders, the difference shows when CEA trust becomes 100 percent. If this is the case, there the initial trust values become absolute so one of the prospects is the absolute best leader. This will lead to either one or (because of a stronger pool) very few leaders.

trust threshold for subscribing to a leader: this variable is constitutes the amount of trust a leader and it’s pool need to have for a client to subscribe. A higher threshold means less clients and thus a negative relation. No threshold means a possible increase in the number of leaders, a 100 percent threshold means no leaders.

trust threshold for changing to a new leader: when a client gets an initiation from a different leader it compares the two. If the new leader offers enough of an increase in trust, it will change to the new leader. When this threshold decreases a greater number of clients will shift leaders possibly creating a more equal distribution of clients and thus possibly, more leaders. This is a negative relation. If there is no threshold, there might be more leaders, if there is a threshold of 100 percent, there will be either one or very few leaders as the leader that first invites clients will collect most of the clients, leaving few for the other prospects.

Minimal number of Clients: the relation of this variable to the number of leaders is straightforward: more clients needed, less leaders, hence a negative relation. When there is no minimum number of clients the number of leaders will increase possibly up to the number of prospects, the minimum number of clients is 100 percent, there will be no leaders.

2.4 Number of Pool members per Leader

The number of pool members per leader is influenced by some of the variables that were discussed during the discussion of the performance indicator number of leaders. The reaction of the number of poolmembers to these variables is similar as an increase in the number of pool members leads to a possible increase in the number of leaders, their behavior in extreme conditions is also similar. Therefore the following variables will not be explicitly discussed in this section but will be shown in the summarizing table at the end of the structural validation:

- max nr of poolmembers
- free capacity required
- free capacity

nr of prospects - nr of leaders: if a number of prospects do not become leader they themselves and their pools will possibly be added to the pools of the leaders that remain, increasing the number of pool members per leader. So if the ratio of nr of prospects/nr of leaders grows, so will the number of pool members per leader, a positive relation. For the extreme condition zero all prospects will have become leaders, leading to a possible decrease, depending on the default situation, in the number of pool members per leader. The extreme condition 100% in this case means that no prospects have become leaders, leading to no poolmembers.

2.5 Number of Clients per leader

As with the number of poolmembers the number of clients per leader has some of its variables in common with the number of leaders. For reasons explained above these variables will not be discussed further:

- initial trust values
- trust in CEA’s
- trust threshold for subscribing to a leader

Number of CE’s: when the number of CE’s increases the number of possible clients does as well, hence this relation is positive. when there are no CE’s there will be no clients, the opposite extreme condition, an infinite number of CE’s cannot be tested.

Number of Leaders: the number of leaders in a network influences the number of clients per leader since the CE’s available to become clients are finite. The relation is negative because more leaders yield less clients per leader. No leaders means no clients, 100 percent leaders is impossible (leaders need clients and pool members to exist).

Number of Poolmembers: the number of poolmembers per leader has a negative influence on the number of clients especially in smaller networks for it adversely affects the number of possible clients (CE’s without a function). No poolmembers would also mean no leaders and thus no clients. 100 percent pool members per leader is equal to the maximum number of pool members allowed per leader and would lead to a slight decrease in the number of CE’s per leader.

2.6 Nr of Pool members with multiple leaders

Pool members can be invited by multiple leaders. If they have sufficient free capacity they can become poolmembers of more than one pool. This shows that the number of pool members with multiple leaders is increased when leaders invite more identical poolmembers, and they, in their turn have more capacity available. This leads to the following variables influencing this performance indicator.

free capacity: this variable has been covered before, it is relation to this particular indicator is positive. In the extreme case of no free capacity there will be no pool members with multiple leaders (actually there will be no pool members at all), in case of 100 percent free capacity there will probably be more.

free capacity required: this was also covered earlier it has a negative relation and acts exactly opposite to free capacity in its extreme conditions.

number of CE’s: the number of CE’s was also covered earlier as a variable but it acts somewhat differently for this indicator. When the number of CE’s increases the range of possible poolmembers grows as well. Hence the number of pool members with multiple leaders will probably decrease: a negative relation. No CE’s leads to no pool members, the opposite cannot be tested.

number of leaders: when the number of leaders increases so does the possibility of leaders picking identical poolmembers, if they have sufficient capacity the number of pool members with multiple leaders will increase. This is a positive relation. No leaders means no pool members and all leaders is impossible.

number of pool members per leader: more pool members per leader means a higher likelihood that pool members are present in multiple pools, a positive relation. As for the extreme cases: if the number of pool members per leader is equal to the maximum number allowed for all pools the indicator might increase further, but if there are no pool members the indicator will be zero.

spread of trust values: this variable has not been encountered before, it basically specifies the level of difference between “good
and bad” CE’s. If the top CE’s are easily distinguishable more leaders will pick them, which is a positive relation. The extreme condition for this factor could not be performed in the simulation and will be not tested.

**Percentage of candidate list that becomes prospect:** the candidate list is not identical for each candidate. The closer one gets to the end of the list, the more the candidate lists will differ. A number of candidates that did not become a prospect can be invited as pool member by other prospects, if their weighted trust value is close enough to the top prospects’. So when a smaller percentage of the candidate list becomes prospect, pool members will be invited from higher up the candidate list, where the lists are more similar, which could lead to a higher score for this indicator. The relation described here is a negative one. If no candidate on the list would become prospect, there would be no leader, and thus no pool members. If all candidates were to become prospects, no pool members would be invited off the candidate lists (they would be invited of the prospects’ weighted Idlists’ which differ a lot) and thus the number of pool members with multiple leaders would decrease.

**Percentage a candidates’ weighted trust value can differ from the top prospects to be invited:** as stated in the discussion of the previous variable the number of candidates that are invited of the candidate list increases the number of poolmembers with multiple pools. Therefore if a candidate’s weighted trust value can differ more from the top prospects weighted trustvalue to still warrant an invitation this performance indicator will increase, making this relation a positive one. If none are invited of the list of remaining candidates the number of pool members with multiple relations will decrease. If all are invited, it will increase.

### 2.7 Results

The table on the following page shows a graphical summary of the relations between certain variables and the relevant performance indicators, discussed in the pervious text. It also contains the simulation results, for the various structural validation tests. These results will be discussed here. Also, possible differences between the expected value and the value obtained through testing will be specified and analyzed.

The table shows all performance indicators and the variables influencing them. The first to columns (reaction indicator on) show the expected behavior of the performance indicator when the variable increases (first column, rise) or decreases (second column, fall). An up arrow indicates that the performance indicator is expected to increase while a down arrow signifies the exact opposite. The third and forth column (xtrm condition) show similar expectancies for the extreme conditions 0 and 100 percent. The results of the structural validation tests are found in the next four columns. The default result for the performance indicator without any changes to one of the variables is shown right after the performance indicator between brackets (number of candidates (15)). The results in the columns five to eight can be compared to this default value to see whether an increase or decrease has occurred for the respective performance indicator.

For the first two indicators: **number of candidates** and **number of prospects**, the results obtained through testing adhere to the expected results of Medusa. The next indicator, **number of leaders**, shows quite some difference between the expected values, and the ones obtained through testing. When the test results are studied more closely it shows that often when an increase is expected, the number of leaders stays the same, four. It was already stated while discussing the expected relations for **number of leaders** that due to the large number of variables it would be possible that a variable, when increased or decreased, would not have a visible effect on the number of leaders. However the fact that in this case the indicator so often does not change only when an increase is expected leads us to believe that there is a specific reason for this behavior. After closely inspecting the default situation the answer became clear. In the default situation the **number of leaders** is already equal to the **number of prospects** (which serves as a cap for the **number of leaders**). So in this situation only a variable that increases the number of prospects can lead to an increase in the number of leaders. So, with this extra observation on the default situation, the results of the simulation do comply with the expected Medusa behavior for the performance indicator: **number of leaders**.

The indicators **number of poolmembers** and **number of clients per leader** correspond nicely to the results expected from Medusa. The few dissimilarities in **number of poolmembers** (an indicator not changing when change is was expected) are due to the fact that any change in the **number of poolmembers** is often a fairly large relative change, too large for the respective variable.

Looking at the indicator **number of poolmembers with multiple pools** one observation stands out. The results of the simulation show a lot of zeros. This however is easily explained. The value of the indicator in the default situation is one. Since this indicator is an integer value, any decrease to the indicator, would lead to a value of zero.

It can be concluded that the simulation has passed all of the structural validation tests and that its results for the tested performance indicators can be considered valid.
<table>
<thead>
<tr>
<th>Expectation</th>
<th>xtrm condition</th>
<th>Results</th>
<th>xtrm condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>indicator reaction</td>
<td>rise</td>
<td>fall</td>
<td>0%</td>
</tr>
<tr>
<td>number of candidates</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of CE's</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>initial trust values</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>free capacity</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>free capacity threshold</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>desire to become leader</td>
<td>↑</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>candidacy threshold</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>number of prospects</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nr of candidates</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>candidate-prospect percentage</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>number of leaders</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of prospects</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>min nr of poolmembers</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>max nr of poolmembers</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>free capacity required</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>initial trust values</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>trust in CEA's</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>trust threshold for subscribing</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>trust threshold for changing</td>
<td>↓</td>
<td>↑</td>
<td>↓</td>
</tr>
<tr>
<td>min nr of clients</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>number of plmbrs per ldr (3,5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max nr of poolmembers</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>free capacity required</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>free capacity</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>nr of prospects - nr of ldrs</td>
<td>↑</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>number of clients per ldr (12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initial trust values</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>trust in CEA's</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>trust threshold for subscribing</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>number of CE's</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>number of leaders</td>
<td>↓</td>
<td>↑</td>
<td>0</td>
</tr>
<tr>
<td>number of plmbrs with multiple pools (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of CE's</td>
<td>↓</td>
<td>↑</td>
<td>0</td>
</tr>
<tr>
<td>spread of trust values</td>
<td>↑</td>
<td>↓</td>
<td>x</td>
</tr>
<tr>
<td>number of leaders</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>free capacity required</td>
<td>↓</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>free capacity</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>candidate-prospect percentage</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
<tr>
<td>percentage of highest tval</td>
<td>↑</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>nr of pool members per leader</td>
<td>↑</td>
<td>↓</td>
<td>0</td>
</tr>
</tbody>
</table>
Appendix D: Scalability Performance Graphs

In this appendix two graphs are presented that give a graphical overview of the scalability performance results in the main text. Both graphs presented here show the number of messages traveling over the average backbone link over time (unquantified). The first graph shows what parts of the graph constitute the various subphases of the bootstrapping phase (primary elections, pool formation, ratification). The second graphs combines the results of 20, 40, 60, 80 and 100 agents in one graphs to show the effects on the number of messages as the number of agents increases.

Figure 4: Subphases Overview

Figure 5: Scalability Performance
Appendix E: Casper Code

In this Appendix the Casper code used for the CSP/FDR analysis is shown. All scenarios and both the original and improved protocols are shown. In the actual Casper code only one protocol and one scenario would be used for a Casper test.

1. PRIMARY ELECTIONS

#Free variables
a, b : Agent
na, nb : Nonce
ts1, ts2 : TimeStamp
PK : Agent -> PublicKey
SK : Agent -> SecretKey
m1 : Cand
m2 : Votes
h : HashFunction
InverseKeys = (PK, SK)

#Processes
INITIATOR(a, m1, na) knows SK(a), PK, h
RESPONDER(b, m2, nb) knows SK(b), PK, h

protocol description
0.   --> a : b
1.  a -> b : {a, na, m1}{SK(a)},{h(m1,a,na),ts1}{SK(a)}
2.  b -> a : {h(m1,a,na,nb),ts1}{SK(b)}, nb
3.  b -> a : {b,nb,{m2}{SK(b)}}{PK(a)},{h(m2,b,nb),ts2}{PK(a)}
4.  a -> b : {h(m2,b,nb,na),ts2}{SK(a)}, na

0.   --> a : b
1.  a -> b : {a, na, m1}{SK(a)},{h(m1,a,na)}{SK(a)}
2.  b -> a : {m2,b,nb,a,na}{PK(a)},{h(m2,b,nb,a,na)}{SK(b)}
3.  b -> a : nb

#Specification
Agreement(a, b, [])
Secret(b, m2, [a])
Agreement(b, a, [])

#Actual variables
TimeStamp = 0 .. 0
MaxRunTime = 0
Alice, Bob, Ivo : Agent
N1, N2, N3: Nonce
M1, M4 : Cand
M2, M3 : Votes

#Inline functions
symbolic PK, SK

#System
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, M2, N2)
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, M2, N2)
INITIATOR(Anne, M1, N1)
INITIATOR(Bob, M4, N2)
RESPONDER(Anne, M2, N3)
RESPONDER(Bob, M2, N2)
RESPONDER(Bob, M3, N3)
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, M2, N2)
RESPONDER(Bob, M3, N3)
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, M2, N2)
RESPONDER(Bob, M3, N3)
INITIATOR(Anne, M1, N1)
INITIATOR(Anne, M1, N2)
RESPONDER(Bob, M2, N3)
RESPONDER(Bob, M2, N3)

#Intruder Information
Intruder = Ivo
IntruderKnowledge = (Alice, Bob, Ivo, PK, SK(Ivo))

Scenario 1: An Initiator Anne, and a responder Bob
Scenario 2: An Initiator Anne, and a responder Anne
Scenario 3: An Initiator Anne, and a responder Bob
Scenario 4: An Initiator Anne, a responder Anne, and a responder Bob
Scenario 5: An Initiator Anne, and two responders Bob
Scenario 6: Two Initiators Anne, and a responder Bob
2. **POOL FORMATION**

# Free variables

\( a, b : \text{Agent} \)

\( na, nb : \text{Nonce} \)

\( ts1, ts2 : \text{TimeStamp} \)

\( PK : \text{Agent} \to \text{PublicKey} \)

\( SK : \text{Agent} \to \text{SecretKey} \)

\( m1 : \text{Covenant} \)

\( h : \text{HashFunction} \)

\( InverseKeys = (PK,SK), (kab,kab) \)

# Processes

\( \text{INITIATOR}(a, m1, na, kab) \text{ knows } SK(a), PK, h \)

\( \text{RESPONDER}(b, nb) \text{ knows } SK(b), PK, h \)

# Protocol description

\( 0. \quad \rightarrow a : b \)

\( 1. \quad a \rightarrow b : \{a,na,kab,\{m1\}\{SK(a)\}\{PK(b)\},h(m1,a,na),ts1\}\{PK(b)\} \)

\( 2. \quad b \rightarrow a : \{h(m1),kab,a,na,rb,tsi\}\{SK(b)\}, nb \)

\( 3. \quad b \rightarrow a : \{h(m1)\}\{SK(b)\},b,nb\{kab\},\{h(m1,b,nb),ts2\}\{kab\} \)

\( 4. \quad a \rightarrow b : \{h(m1,b,nb,na),ts2\}\{kab\},na,b \)

\( 0. \quad \rightarrow a : b \)

\( 1. \quad a \rightarrow b : \{a,na,kab,\{m1\}\{PK(b)\},\{h(a,kab,m1)\}\{SK(a)\}\} \)

\( 2. \quad b \rightarrow a : \{h(m1)\}\{SK(b)\},b,nb\{kab\},\{h(m1,b,nb)\}\{kab\} \)

\( 3. \quad a \rightarrow b : nb \)

# Specification

Agreement(a, b, [])

Secret(a, m1, [b])

Secret(a, kab, [b])

Agreement(b, a, [])

# Actual variables

TimeStamp = 0 .. 0

MaxRunTime = 0

Anne, Bob, Ivo : Agent

N1, N2: Nonce

M1, M2 : Covenant

Kab, Kcd : SessionKey

InverseKeys = (Kab,Kab) (Kcd,Kcd)

# Inline functions

symbolic PK, SK

# System

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{RESPONDER}(Bob, N2) \)

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{RESPONDER}(Anne, N2) \)

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{INITIATOR}(Bob, M2, N2, Kcd) \)

\( \text{RESPONDER}(Anne, N3) \)

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{RESPONDER}(Anne, N2) \)

\( \text{RESPONDER}(Bob, N3) \)

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{RESPONDER}(Bob, N2) \)

\( \text{RESPONDER}(Bob, N3) \)

\( \text{INITIATOR}(Anne, M1, N1, Kab) \)

\( \text{INITIATOR}(Anne, M2, N2, Kcd) \)

\( \text{RESPONDER}(Bob, M2, N3) \)

# Intruder Information

Intruder = Ivo

IntruderKnowledge = (Alice, Bob, Ivo, PK, SK(Ivo))
3. RATIFICATION

#Free variables
a, b : Agent
na, nb : Nonce
ta1, ta2, ta3 : TimeStamp
PK : Agent -> PublicKey
SK : Agent -> SecretKey
m1 : Inv
m2 : Secr
kab : SessionKey
h : HashFunction
InverseKeys = (PK, SK), (kab, kab)

#Processes
INITIATOR(a, m1, na, kab) knows SK(a), PK, h
RESPONDER(b, m2, nb) knows SK(b), PK, h

#Protocol description
0.  -> a : b
1.  a -> b : {a, na, m1} {SK(a)}, {h(m1, a, na, ta1)} {SK(a)}
2.  b -> a : {h(m1, a, na, nb), ta1} {SK(b)}, nb
3.  a -> b : {h(m1, a, na, nb)} {SK(a)}
4.  b -> a : {h(m2, b, nb), ta2} {PK(a)}, nb
5.  a -> b : {a, na, m2, Kab} {PK(b)}, {h(m2, a, na, ta3)} {PK(b)}
6.  b -> a : {h(m1, a, na, nb, ta3)} {kab}, nb

0.  -> a : b
1.  a -> b : {a, na, m1} {SK(a)}, {h(m1, a, na)} {SK(a)}
2.  b -> a : {m2, a, na, nb} {PK(a)} {h(m2, a, na, nb)} {SK(b)}
3.  a -> b : {a, m2, Kab, nb} {PK(b)}, {h(m2, a, kab, nb)} {SK(a)}

#Specification
Agreement(a, b, [])
Secret(b, m2, [a])
Secret(a, kab, [b])
Agreement(b, a, [])

#Actual variables
TimeStamp = 0 .. 0
MaxRunTime = 0
Anne, Bob, Ivo : Agent
N1, N2, N3: Nonce
M1 : Inv
Kab, Kcd : SessionKey
M2, M3 : Secr
InverseKeys = {Kab, Kab}, (Kcd, Kcd)

#Inline functions
symbolic PK, SK

#System
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Bob, M2, N2)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M2, N2)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Bob, M1, N2, Kcd)
RESPONDER(Anne, M2, N3)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M2, N2)
RESPONDER(Bob, M3, N3)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Bob, M2, N2)
RESPONDER(Bob, M3, N3)
INITIATOR(Anne, M1, N1, Kab)
INITIATOR(Anne, M1, N2, Kcd)
RESPONDER(Bob, M2, N3)

#Intruder Information
Intruder = Ivo
IntruderKnowledge = {Anne, Bob, Ivo, PK, SK(Ivo)}

Scenario 1: An Initiator Anne, and a responder Bob
Scenario 2: An Initiator Anne, and a responder Anne
Scenario 3: An Initiator Anne, a responder Anne, and an initiator Bob
Scenario 4: An Initiator Anne, a responder Anne, and a responder Bob
Scenario 5: An Initiator Anne, and two responders Bob
Scenario 6: Two Initiators Anne, and a responder Bob
4. TRUST TOKEN SHARING

#Free variables
a, b : Agent
na, nb : Nonce
ts1 : TimeStamp
m1 : Trust
h : HashFunction
SK : Agent -> ServerKey
InverseKeys = (SK, SK)

#Processes
INITIATOR(a, m1, na) knows SK, h
RESPONDER(b, nb) knows SK(b), h

#Protocol description
0.    -> a : b
1.  a -> b : {a, b, na, m1}{SK(b)}, {h(m1, a, na), ts1}{SK(b)}, nb
2.  b -> a : {h(m1, a, na, nb), ts1}{SK(b)}
3.  a -> b : nb

#Specification
Agreement(a, b, [])
Secret(b, m1, [a])
Agreement(b, a, [])

#Actual variables
TimeStamp = 0 .. 0
MaxRuntime = 0
Anne, Bob, Ivo : Agent
N1, N2, N3: Nonce
M1, M2 : Trust

#Inline functions
symbolic SK

#System
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, N2)
INITIATOR(Anne, M1, N1)
RESPONDER(Anne, N2)
INITIATOR(Anne, M1, N1)
INITIATOR(Bob, M2, N2)
RESPONDER(Anne, N3)
INITIATOR(Anne, M1, N1)
RESPONDER(Anne, N2)
RESPONDER(Bob, N3)
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, N2)
RESPONDER(Bob, N3)
INITIATOR(Anne, M1, N1)
INITIATOR(Anne, M2, N2)
RESPONDER(Bob, N3)

#Intruder Information
Intruder = Ivo
IntruderKnowledge = (Anne, Bob, Ivo)

Scenario 1: An Initiator Anne, and a responder Bob
Scenario 2: An Initiator Anne, and a responder Anne
Scenario 3: An Initiator Anne, and a responder Bob
Scenario 4: An Initiator Anne, a responder Anne, and a responder Bob
Scenario 5: An Initiator Anne, and two responders Bob
Scenario 6: Two Initiators Anne, and a responder Bob
5. ALIVE

#Free variables
a, b : Agent
na, nb : Nonce
ts1 : TimeStamp
m1 : Alve
h : HashFunction
SK : Agent -> ServerKey
InverseKeys = (SK, SK)

#Processes
INITIATOR(a, m1, na) knows SK, h
RESPONDER(b, nb) knows SK(b), h

#Protocol description
0.    -> a : b
1.  a -> b : {a, na, m1}{SK(ab)}, {h(m1, a, na), ts1}{SK(ab)}
2.  b -> a : {h(m1, a, na, nb, ts1)}{SK(ab)}, nb

0.    -> a : b
1.  a -> b : {a, m1}{SK(ab)}, {h(m1, a), ts1}{SK(ab)}

#Specification
Aliveness(a, b)

#Actual variables
TimeStamp = 0 .. 0
MaxRunTime = 0
Anne, Bob, Ivo : Agent
N1, N2, N3: Nonce
M1 : Alve

#Inline functions
symbolic SK

#System
INITIATOR(Anne, M1, N1)
RESPONDER(Bob, N2)

INITIATOR(Anne, M1, N1)
RESPONDER(N2)

INITIATOR(Anne, M1, N1)
INITIATOR(Bob, M2, N2)
RESPONDER(N3)

INITIATOR(Anne, M1, N1)
RESPONDER(N2)
RESPONDER(N3)

INITIATOR(Anne, M1, N1)
RESPONDER(N2)
RESPONDER(N3)

INITIATOR(Anne, M1, N1)
INITIATOR(Bob, M2, N2)
RESPONDER(N3)

INITIATOR(Anne, M1, N1)
INITIATOR(Bob, M2, N2)
RESPONDER(N3)

INITIATOR(Anne, M1, N1)
INITIATOR(Bob, N3)
RESPONDER(N3)

#Intruder Information
Intruder = Ivo
IntruderKnowledge = {Anne, Bob, Ivo}

---

Original Protocol
Original Protocol
Original Protocol

Improved Protocol
Improved Protocol
Improved Protocol

Scenario 1: An Initiator Anne, and a responder Bob
Scenario 2: An Initiator Anne, and a responder Anne
Scenario 3: An Initiator Anne, and a responder Bob
Scenario 4: An Initiator Anne, a responder Anne, and a responder Bob
Scenario 5: An Initiator Anne, and two responders Bob
Scenario 6: Two Initiators Anne, and a responder Bob
6. RESURRECTION

# Free variables
a, b: Agent
s: Server
na, nb, nc: Nonce
t1, t2, t3, t4, t5: TimeStamp
m1: DD
m2: TT
h: HashFunction
PK: Agent -> PoolKey
SK: Agent -> ServerKey
InverseKeys = (SK, SK), (PK, PK)

# Processes
INITIATOR(a, s, m1, m2, na) knows SK(a), PK(a), h
RESPONDER(b, s, m2, nb) knows SK(b), PK(s), h
SERVER(s, nc) knows SK, h

# Protocol description

0. -> a : b
1. a -> b : {a, na, m1}{PK(a)}, {h(m1, a, na), ts1}{PK(a)}
3. a -> s : {a, na, m1}{SK(a)}, {h(m1, a, na), ts2}{SK(a)}
4. b -> a : {h(m1, a, na, nb), ts1}{PK(a)}, nb
5. b -> s : {h(m1, a, na, nb), ts3}{SK(b)}, nc
6. s -> a : {h(m1, a, na, nc), ts2}{SK(a)}, nc
7. s -> b : {h(m1, a, na, nc), ts3}{SK(b)}, nc
8. a -> s : {a, na, m2}{SK(a)}, {h(m2, a, na), ts4}{SK(a)}
9. b -> s : {b, m2, nb}{SK(b)}, {h(m2, b, nb), ts5}{SK(b)}
10. s -> a : {h(m2, a, na, nc), ts4}{SK(a)}, nc
11. s -> b : {h(m2, b, nb, nc), ts5}{SK(b)}, nc

0. -> a : b
1. a -> b : {a, na, m1}{PK(a)}, {h(m1, a, na)}{PK(a)}
3. a -> s : {a, na, m1}{SK(a)}, {h(m1, a, na)}{SK(a)}
4. b -> a : {h(m1, a, na, b)}{PK(a)}, b
5. b -> s : {h(m1, a, na, b)}{SK(b)}, b
6. s -> a : {h(m1, a, na, s)}{SK(a)}, s
7. s -> b : {h(m1, a, na, s)}{SK(b)}, s
8. a -> s : {a, na, m2}{SK(a)}, {h(m2, a, na)}{SK(a)}
9. b -> s : {b, m2, nb}{SK(b)}, {h(m2, b, nb)}{SK(b)}
10. s -> a : na
11. s -> b : nb

# Specification
Secret(a, m2, [b, s])
Agreement(a, b, [])
Agreement(b, a, [])
Agreement(a, s, [])
Agreement(s, a, [])
Agreement(s, b, [])
Agreement(b, s, [])

# Actual variables
TimeStamp = 0 .. 0
MaxRunTime = 0
Anne, Bob, Ivo: Agent
Jeeves: Server
N1, N2, N3, N4: Nonce
M1: DD
M2, M3, M4: TT

# Inline functions
symbolic SK, PK

# System
INITIATOR(Anne, M1, M2, N1)
RESPONDER(Bob, M3, N2)
SERVER(Jeeves, N3)
INITIATOR(Anne, M1, N2, N1)
RESPONDER(Anne, M2, N2)
SERVER(Jeeves, N3)
INITIATOR(Anne, M1, M2, N1)
INITIATOR(Bob, M1, M3, N2)
RESPONDER(Anne, M4, N3)
SERVER(Jeeves, N4)

Scenario 3: An Initiator Anne, a responder Anne, and an initiator Bob

INITIATOR(Anne, M1, M2, N1)
RESPONDER(Anne, M3, N2)
RESPONDER(Bob, M4, N3)
SERVER(Jeeves, N4)

Scenario 4: An Initiator Anne, a responder Anne, and a responder Bob

INITIATOR(Anne, M1, M2, N1)
RESPONDER(Bob, M3, N2)
RESPONDER(Bob, M4, N3)
SERVER(Jeeves, N4)

Scenario 5: An Initiator Anne, and two responders Bob

INITIATOR(Anne, M1, M2, N1)
INITIATOR(Anne, M1, M3, N2)
RESPONDER(Bob, M4, N3)
SERVER(Jeeves, N4)

Scenario 6: Two Initiators Anne, and a responder Bob

#Intruder Information
Intruder = Ivo
IntruderKnowledge = {Anne, Bob, Ivo, Jeeves}
7. KEY & SECRET REFRESHMENT

#Free variables
a, b : Agent
na, nb : Nonce
m1, m2 : Secr
h : HashFunction
SK : Agent -> ServerKey
kab : SessionKey
InverseKeys = (SK, SK), (kab, kab)

#Processes
INITIATOR(a, m1, na, kab) knows SK, h
RESPONDER(b, m1, m2, nb) knows SK(b), h

#Protocol description
0.   -> a : b
1.   a -> b : {a, b, na, kab, m1}{SK(b)}, {h(m1, a, na), ts1}{SK(b)}
2.   b -> a : {h(m1, a, na, nb, kab), nb}{SK(b)}
3.   b -> a : {b, nb, m2}{kab}, {h(m2, b, nb)}{kab}
4.   a -> b : {h(m2, b, nb, na)}{kab}, na

0.   -> a : b
1.   a -> b : {kab, m1, a}{SK(b)}, {h(a, m1)}{SK(b)}
2.   b -> a : {b, nb, m2}{kab}, {h(m2, b, nb)}{kab}
3.   a -> b : nb

#Specification
Agreement(a, b, [])
Secret(b, m2, [a])
Secret(a, kab, [b])
Agreement(b, a, [])

#Actual variables
Anne, Bob, Ivo : Agent
N1, N2, N3 : Nonce
Kab, Kcd : SessionKey
M1, M2 : Secr
InverseKeys = (Kab, Kab), (Kcd, Kcd)

#Inline functions
symbolic SK

#System
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Bob, M1, M2, N2)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M1, M2, N2)
INITIATOR(Anne, M1, N1, Kab)
INITIATOR(Bob, M1, N2, Kcd)
RESPONDER(Anne, M1, M2, N3)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M1, M2, N2)
RESPONDER(Anne, M1, M2, N3)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M1, M2, N2)
RESPONDER(Anne, M1, M3, N3)
INITIATOR(Anne, M1, N1, Kab)
RESPONDER(Anne, M1, M2, N2)
RESPONDER(Anne, M1, M3, N3)
RESPONDER(Anne, M1, M2, N3)
INITIATOR(Anne, M1, N1, Kab)
INITIATOR(Anne, M1, N2, Kcd)
RESPONDER(Anne, M1, M2, N3)
RESPONDER(Anne, M1, M2, N3)

#Intruder Information
Intruder = Ivo
IntruderKnowledge = {Anne, Bob, Ivo}
References


[33] [www.formal.demon.uk](http://www.formal.demon.uk)

[34] [www.forrester.com](http://www.forrester.com)

[35] [www.isi.edu/nsnam/ns/](http://www.isi.edu/nsnam/ns/)

[36] [www.rhk.com](http://www.rhk.com)

[37] [www.yankeegroup.com](http://www.yankeegroup.com)