Validating the Security of Medusa: A survivability protocol for security systems

Wiebe Wiechers, Semir Daskapan

Faculty Technology, Policy and Managment, Delft University of Technology, Delft, Netherlands

Abstract. In this paper a new approach for enabling survivable secure communications in multi agent systems is validated through CSP/FDR state analysis. 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 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.

1 Introduction

Traditionally, the trust management service in most networks is provided by a centralized authority: the Security Distribution Center (SDC). This centralized approach (Steiner, 1988) brings advantages such as efficiency and trustworthiness (an agent only has to trust the centralized entity), which are lost in decentralized [2] and distributed [4] [20] [21] 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 centres. 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 [7]. 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 [6]. In Medusa this moveable software object, called the trust token, is moved to another location when the SDC goes down, so that the service can be continued. To that end, Medusa creates a pool of possible successors to the current SDC. In theory, this aspect of Medusa

increases the survivability of the SDC indefinitely, because Medusa uses resource sharing between undedicated systems. 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 [10].
Medusa's ability to provide enhanced survivability was already proven in earlier research [19] however for Medusa to be successful it must also be secure. In this paper the security of the Medusa process is put to the test.

In the discussion of related works two types of works are distinguished. First, there are works related to Medusa. Group Communication Systems (GCS) [1] [14] also offer a trusted infrastructure and some [11] [13] [3], like Medusa, are specifically designed to withstand failures and malicious agents, however these GCS's feature an unranked infrastructure. A Medusa-like hierarchical structure is proposed in [11], but it does not achieve survivability through the use of undedicated systems and resource sharing.

Second, there are works similar to this specific paper, a security validation using the CSP/FDR approach. A similar approach is used in [9] for validating existing protocols, but not for Medusa.

Medusa is designed to cover both internal security (systems security) and external security (communications 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 [16].

In section 2 an overview is given of Medusa's security requirements, the communication during its lifecycle, together with a logical analysis of the security requirements. In section 3, the CSP/FDR state enumeration approach is introduced and used to find any possible attacks an intruder could use to subvert Medusa's requirements. Also, this approach is used to see whether the security properties can be asserted more efficiently. Finally, the results of these analyses are summarized and discussed further in section 4.

2 MEDUSA: Security Requirements & Overview

First the three security requirements are defined. Following Medusa is introduced and its communication evaluated. Each security requirement is analyzed to see whether it is required and delivered (in a friendly environment).

2.1 Security Requirements

Although Medusa was designed as a single solution for security services, it consists of multiple security protocols occurring either sequentially or concurrently. A security protocol is defined as: a prescribed sequence of interactions between entities to achieve a certain end [16]. 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 modelled 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.

The three security properties that are 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:* if a message alleges to be from a certain entity, it was indeed originated by that entity.

Of these security requirements 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 message. For authentication two forms are distinguished: one-way authentication, signifying that the initial sender must be authenticated to the receiver, and two-way authentication meaning that the receiver should also be authenticated to the sender.

2.2 Medusa Overview

The Medusa process consists of three phases: bootstrapping, preparation and resurrection. The preparation and resurrection phases are part of Medusa’s operational lifecycle. The preparation phase ensures the survival of the ad-hoc SDC, through continuous replication and distribution of its trust token. When the SDC goes down the resurrection phase begins and another SDC is established by reconstructing the trust token. 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.

In the following sections an overview is presented of the different phases of Medusa. The various security protocols taking place are shown with their sequence of messages. For coherency with the Medusa pseudo code the methods in which the messages are sent are shown as well. For every method the necessary security requirements are defined. By looking at the various security tools used in the protocols it is determined if and how these requirements are met. Here is an overview of the various security tools used in Medusa communication and their abbreviations:

A,B : identity of a or b	*K_{ab} : symmetric key shared by A and B*
M : the actual message	*$\{m\}SK_a$: message encrypted with SK_a*
N_a : nonce generated by a	*TS : timestamp*
	H(M) : a hash of M

PK_a , SK_a : public and secret key pair

2.2.1 The Bootstrapping Phase

The bootstrapping phase itself can be further divided into three subphases: Primary Elections, Pool Formation, Ratification.

Primary Elections

Before bootstrapping can begin, a network must exist with a number of agents (CE’s). 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), i.e a trust dealer. A CEA 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. Agents that opt to become a candidate leader inform 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. After receiving all the votes the candidates individually calculate the score of each candidate by weighting the votes of every agent with their 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.

1. CE $\rightarrow$ CEA :	(IDlist request)	*Initialize*
2. CEA $\rightarrow$ CE :	(IDlist)	
3. Cand(A)$\rightarrow$CE(B): $\{M,A,N_a\}SK_a$, $\{H(M,A,N_a),TS\}SK_a$	(candidacy declaration)	*ModPublishCand*
4. CE(B) $\rightarrow$ Cand(A): $\{H(M,A,N_a,N_b),TS\}SK_b$, N_b	(acknowledge receipt)	
5. CE(B) $\rightarrow$ Cand(A): $\{M,B,N_b\}PK_a$, $\{H(M,B,N_b),TS\}PK_a$	(ballot list)	*ModCastBallots*
6. Cand(A) $\rightarrow$ CE(B): $\{H(M,B_b,N_a),TS\}SK_a$, N_a	(acknowledge receipt)	

Initialize: In this initial protocol every CE acquires an initial IDlist from the CEA containing the ID’s and trust certificates of most CE’s in the network space. Correct authentication both from the CE to the CEA and from the CEA to the CE is essential in the method. 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. 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 that a CEA in the Medusa environment has some protocol in place which offers confidentiality and 2-way authentication.

ModPublishCand: In this method the candidates broadcast their candidacy declaration over the network. Since the candidacy declaration does not contain any sensitive information confidentiality of the message is not an issue here. Authentication however is relevant; and is provided through the use of the Candidate’s secret key.

ModCastBallots: A CE sends its list of ballots to all known candidates. Similar to the method Initialize confidentiality is an issue here for reasons of privacy. It is provided by encrypting the message with the candidate’s public key. Authentication is also required but is <u>not</u> provided, anybody could have encrypted this message since the key used for encryption is public.

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. The invited pool members evaluate whether to indeed become a pool member of the inviting prospect. They must have sufficient trust in the inviting prospect to do so. Once enough pool members agree the pool formation is finished.

$$\left.\begin{array}{ll}1.\ \text{Pros}(A)\rightarrow\text{CE}(B): \{\{M\}SK_a,A,N_a,K_{ab}\}PK_b,\{H(M,A,N_a)TS\}PK_b & \text{(Covenant)}\\ 2.\ \text{CE}(B)\rightarrow\text{Pros}(A): \{H(M,K_{ab},A,N_a,N_b),TS\}SK_b,\ N_b & \text{(Acknowledge Receipt)}\end{array}\right\}\textit{ModProcessVotes}$$

$$\left.\begin{array}{ll}3.\ \text{CE}(B)\rightarrow\text{Pros}(A): \{\{H(M)\}SK_b,A,N_b\}K_{ab},\{H(M,A,N_b)TS\}K_{ab} & \text{(Signed covenant)}\\ 4.\ \text{Pros}(A)\rightarrow\text{CE}(B): \{H(\{M\}SK_b,,A,N_b,N_a),TS\}SK_a,\ N_a & \text{(Acknowledge Receipt)}\end{array}\right\}\textit{ModVouches}$$

ModProcessVotes: 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. Confidentiality is provided by encrypting the entire message with the CE's public key. Also, authentication should be established from the prospect to the future pool member. Authenticity is guaranteed because M is signed with the prospect's private key.

ModVouches: In Modvouches the prospect pool member signs and returns the covenant. 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 leader. 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. Both authenticity and confidentiality are achieved by encrypting the message with its symmetric key supplied by the prospect.

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 possible future successors. The agents individually decide whether or not to become a client of a certain leader (or SDC). 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.

$$\left.\begin{array}{ll}1.\ \text{Pros}(A)\rightarrow\text{CE}(B): \{M,A,N_a\}SK_a,\ \{H(M,A,N_a),TS\}SK_a & \text{(Subscr Inivitation)}\\ 2.\ \text{CE}(B)\rightarrow\text{Pros}(A): \{H(M),A,N_a,N_b),TS\}SK_b,\ N_b & \text{(Acknowledge Receipt)}\end{array}\right\}\textit{ModRatification}$$

$$\left.\begin{array}{ll}3.\ \text{CE}(B)\rightarrow\text{Pros}(A):\{\{M\}PK_a,A,N_b\}SK_b,\{H(M,A,N_b),TS\}SK_b & \text{(Subscr Answer)}\\ 4.\ \text{Pros}(A)\rightarrow\text{CE}(B): \{H(M,A,\ N_b,N_a),TS\}SK_a,\ N_a & \text{(Acknowledge Receipt)}\end{array}\right\}\textit{ModSubscribe}$$

$$\left.\begin{array}{ll}5.\ \text{Pros}(A)\rightarrow\text{CE}(B): \{M,K_{ab,}A,N_a\}PK_b,\ \{H(M,A,N_a)TS\}PK_b & \text{(Secret Key)}\\ 6.\ \text{CE}(B)\rightarrow\text{Pros}(A): \{H(M,A,N_a,TS,N_b)\}SK_b,\ N_b & \text{(Acknowledge Receipt)}\end{array}\right\}\textit{ModMembership}$$

ModRatification: As in the method ModPublishCand the leader broadcasts it subscription request over the network, hence 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. This is provided by encrypting the subscription request with the leader's secret key.

ModSubscribe: The CE chooses whether or not to subscribe and replies to the leader. If the client subscribes it sends a hashed secret to the leader which should remain confidential. The CE must also be authenticated to the leader. Confidentiality is achieved by encrypting the whole message with the prospect's public key; the hashed secret is encrypted with the client's private key to provide authenticity.

ModMembership: The leader acknowledges the client's subscription. This 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. Confidentiality is provided 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.

2.2.2 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 secret sharing [18] [5]. 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.

There are two security protocols that run during the preparation phase: the trust token sharing protocol and the alive protocol. The first is run only periodically when a significant change to the secret has occurred (due to for instance a new client being added) while the latter is run almost continuously.

Trust Token Sharing:

$$\left.\begin{array}{lr} 1.\ \mathrm{Ldr}(A) \rightarrow \mathrm{Plmbr}(B): \{M,A,N_a\}K_{ab},\{H(M,A,N_a),TS\}K_{ab} & \text{(piece of token)} \\ 2.\ \mathrm{Plmbr}(B) \rightarrow \mathrm{Ldr}(A): \{H(M,A,N_a,N_b),TS\}K_{ab}, N_b & \text{(Acknowledge Receipt)} \end{array}\right\} \textit{ModPreparation}$$

Alive:

$$\left.\begin{array}{lr} 1.\ \mathrm{Ldr}(A) \rightarrow \mathrm{Plmbr}(B): \{M,A,N_a\}K_{ab}, \{H(M,A,N_a),TS\}K_{ab} & \text{(Alive message)} \\ 2.\ \mathrm{Ldr}(B) \rightarrow \mathrm{Plmbr}(A): \{H(M,A,N_a,N_b),TS\}K_{ab}, N_b & \text{(Acknowledge Receipt)} \end{array}\right\} \textit{ModSelfAssess}$$

ModPreparation: this method handles the division and distribution of the survivability objects. 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. Both authenticity and confidentiality are provided through a symmetric key shared by the leader and the respective pool member.

ModSelfAssess: This method contains a leader's functionality for sending "alive" signals to each pool member. The security requirements for "alive" messages are limited. Confidentiality is not required, 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 an imposter. Authenticity is guaranteed through symmetric keys.

2.2.3 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 of 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 a 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.

The resurrection phase features two security protocols: resurrection and key & secret refreshment. Contrary to the preparation phase these protocols run sequentially.

Resurrection:

1.Plmbr(A) $\rightarrow$ Ldr(B): $\{M,A,N_a\}K_{ab},\{H(M,A,N_a),TS\}\ K_{ab}$ (request Alive)
2.Plmbr(A)$\rightarrow$Plmbr(C):$\{M,A,N_a\}K_{ac},\{H(M,A,N_a),TS\}K_{ac}$ (death decl.)
3.Plmbr (C) $\rightarrow$ Plmbr(A): $\{H(M,A,N_a,N_c),TS\}K_{ac}$, N_c (Acknowledge dd)
} *ModCeckCondition*

4.Plmbr(A)$\rightarrow$Scssr(C):$\{M,A,N_a\}K_{ac}$, $\{H(M,A,N_a),TS\}K_{ac}$ (trust token)
5.Sccssr(C) $\rightarrow$ Plmbr(A): $\{H(M,A,N_a,N_c),TS\}K_{ac}$, N_c (Ackn. receipt)
} *ProcSendToken*

Key & Secret Refreshment

1. Ldr(A) $\rightarrow$ CE(B): $\{H(M),A,N_a,K_{ab2})K_{ab}$, $\{H(K_{ab2},A,N_a),TS\}K_{ab}$ (new key)
2. CE(B) $\rightarrow$ Ldr(A): $\{H(K_{ab2},A,N_a,N_b),TS\}K_{ab}$, N_b (Acknowledge Receipt)
} *ProcRefreshKeys*

3. CE(B) $\rightarrow$ Ldr(A): $\{H(M_2),B,N_b\}K_{ab2}$, $\{H(M_2,B,N_b),TS\}K_{ab2}$ (updated sectret)
4. Ldr(A) $\rightarrow$ CE(B): $\{H(M_2,B,N_a,N_b),TS\}K_{ab2}$, N_b (Acknowledge receipt)
} *ProcRefreshSecr*

ModCheckCondition: After a failed alive check, a poolmember informs the others of the leader's demise and starts the resurrection protocol. Informing the other pool members of the leader's demise can 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. Authenticity is provided through the symmetric keys

ProcSendToken: In this method all pool members send their piece of the token to the successor. Confidentiality is critical since together the pieces of the token contain highly sensitive information. Also two-way authentication must be established. All requirements are achieved by the symmetric keys used in both messages.

ProcRefreshKeys: The successor has assumed the leadership role and refreshes all keys. Since this is in essence a key-exchange protocol again both confidentiality and two-way authentication need to be established. These requirements are provided via the symmetric keys of the former leader which the successor has successfully rebuilt.
ProcRefreshSecr: After receiving a fresh key the CE also resends its secret to the leader. 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 again two-way authentication is required. Both authenticity and confidentiality are provided through the use of the new symmetric keys, just issued by the new leader in procRefreshKeys.

2.3 Summary

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 1. Security Requirements

Methode	Auth.	Int	Con	Methode	Auth.	Int	Con
Initialize	2-way	+	+	ModMembership	1-way	+	+
ModPublishCnd	1-way	+	-	ModPreparation	2-way	+	+
ModCastBallots	1-way	+	+	ModSelfAssess	1-way	+	-
ModProcessVts	1-way	+	+	ModChkCndition	1-way	+	+
ModVouches	1-way	+	+	ProcSendToken	2-way	+	+
ModRatification	1-way	+	-	ProcRefreshKeys	2-way	+	+
ModSubscribe	1-way	+	+	ProcRefreshSecr	2-way	+	+

All messages in the Medusa process include a signed hash of the message contents. Therefore the integrity of all messages is guaranteed. In most cases the other requirements are also met except for one: the authenticity requirement in *modCastBallots*. Authenticity is not achieved because the message is only encrypted using the receiver's public key. If instead the message is also encrypted with the sender's private key, as seen in *modProcessVotes*, both authenticity and confidentiality will be guaranteed as is required.

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 attempt to subvert the Medusa process. To validate Medusa in a hostile environment the CSP/FDR approach is introduced.

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 [12] [15] [17]. 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. The intruder conforms to the Dolev-Yao model [8]. 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.

For generating the CSP language a high level compiler was used: Casper (Lowe, 1998). Casper takes a fairly abstract description of a security protocol (similar to the description used in section 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.

3.1 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. 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. The protocol specifications of section 3 are in several systems, containing various initiators and responders. The following scenarios are a reasonably complete list of the checks worth making [16]:

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 B*
5. *An initiator A, and two responders B*
6. *Two Initiators A, and a responder B*

All of these systems were tested for each Medusa security protocol. The following text describes the attacks that were found on the various protocols and, if possible, a solution. If a protocol is not mentioned, no attack was found.

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

1.Pros(A)$\rightarrow$Int :{{M}SK$_a$,A,N$_a$,K$_{a,int}$}PK$_{int}$,{H(M,A,N$_a$)TS}PK$_{int}$

1.Int(A)$\rightarrow$CE(B):{{M}SK$_a$,A,N$_a$,K$_{a,int}$}PK$_b$,{H(M,A,N$_a$)TS}PK$_b$

2.CE(B)$\rightarrow$Int(A):{H(M, K$_{a,int}$,A,N$_a$,N$_b$),TS}SK$_B$, N$_b$

3.CE(B)$\rightarrow$Int(A):{{H(M)}SK$_b$,A,N$_b$}K$_{a,int}$,{H(M,A,N$_b$)TS}K$_{a,int}$

4.Int(A) $\rightarrow$CE :{H({M}SK$_b$,,A, N$_b$,N$_a$),TS} K$_{a,int}$, N$_a$

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 completely by adding the message's destination in the part encrypted with A's secret key. The protocol is changed as follows: (the change is printed bold):

1. Pros(A)→CE(B):$\{\{M,\mathbf{B}\}SK_a A,N_a K_{ab}\}PK_b\{H(M,A,N_a)TS\}PK_b$
2. CE(B)→Pros(A): $\{H(M,K_{ab},A,N_a,N_b),TS\}SK_b$, N_b
3. CE(B)→Pros(A): $\{\{H(M)\}SK_b,A,N_b\}K_{ab}$, $\{H(M,A, N_b)TS\}K_{ab}$
4. Pros(A)→CE(B): $\{H(\{M\}SK_b,,A,N_b,N_a),TS\}SK_a$, N_a

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

Trust Token-Sharing: In scenario 4 Casper detects the following attack on the authentication of *ModPreparation*:

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

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 A for as long as the timestamp is valid, or saves the last timestamp it has received, this replay attack becomes impossible. The attack can also be solved regardless of implementation by adapting the protocol as follows:

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

The nonce generated by B is sent to A encrypted, and subsequently returned to B. Every time B receives its nonce it can therefore 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 both, strong authentication is guaranteed.

3.2 Protocol Improvements

To optimize the Medusa protocol further, CSP/FDR was used to check the security requirements could still be achieved if the various sub protocols were "slimmed down". 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 greatly benefit to the protocol if the time synchronization dependency were removed.

Primary Elections:

1. Cand(A)$\rightarrow$CE(B) : {M,A,N_a}SK_a, {H(M,A,N_a)}SK_a
2. CE(B)$\rightarrow$Cand(A) : {M,B,N_b,A,N_a}PK_a,{H(M,B,N_b,A,N_a}SK_b
3. Cand(A)$\rightarrow$CE(B) : N_b

Pool Formation:

1. Pros(A)$\rightarrow$CE(B) : {M,,A,K_{ab}}PK_b,{H(M,A,K_{ab}),B}SK_a
2. CE(B)$\rightarrow$Pros(A) : {{H(M)}SK_b,B,N_b}K_{ab},{H(M,B,N_b)}K_{ab}
3. Pros(A)$\rightarrow$CE(B) : N_b

Ratification:.

1. Cand(A)$\rightarrow$CE(B) : {M,A,N_a}SK_a, {H(M,A,N_a)}SK_a
2. CE(B)$\rightarrow$Cand(A) : {M,B,N_b,A,N_a}PK_a,{H(M,B,N_b,A,N_a}SK_b
3. Cand(A)$\rightarrow$CE(B) : {M,A,N_b,K_{ab}} PK_b,{H(M A,N_b,K_{ab})}SK_b

Trust Token Sharing:

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

Alive:

1. Plmbr(A) $\rightarrow$ Ldr(B) : N_b
2. Ldr(A) $\rightarrow$ Plmbr(B) : {M,A, N_b } K_{ab}, {H(M,A, N_b } K_{ab}

1. Ldr(A) $\rightarrow$ Plmbr(B) : {M,A,TS} K_{ab}, {H(M,A,TS} K_{ab}

Resurrection:

1. Plmbr (A) $\rightarrow$ Ldr(B) : {M,A,N_a} K_{ab}, {H(M,A,N_a)} K_{ab}
2. Plmbr (A) $\rightarrow$ Plmbr(C) : {M,A,N_a} K_{ac}, {H(M,A,N_a)} K_{ac}
3. Plmbr (C) $\rightarrow$ Plmbr(A) : {C, H(M,A,N_a,C)}K_{ac}
4. Plmbr (A) $\rightarrow$ Sccssr(C) : {M,A,N_a} K_{ac}, {H(M,A,N_a)} K_{ac}
5. Sccssr (C) $\rightarrow$ Plmbr(A) : N_a

Key & Secret Refreshment:

1. Ldr(A)$\rightarrow$CE(B):{H(M),A,K_{ab2})K_{ab},{H(K_{ab2},A)}K_{ab}
2. CE(B)$\rightarrow$Ldr(A): {H(M_2),B,N_b} K_{ab2}, {H(M_2,B,N_b)} K_{ab2}
3. Ldr(A)$\rightarrow$CE(B): N_b

Comparing these protocols to the original protocols a number of improvements are visible. First and foremost the timestamps have been removed without sacrificing the required security. Any replay attacks are avoided through the use of nonces. Furthermore most individual protocols have become much smaller, with in some cases a 50 percent reduction in the number of messages required.
For the A*live* protocol two possible implementations are shown. The first does not use timestamps but consists of two messages. The second consists of only one message (a 50 percent reduction in the number of messages) but does require the use of a timestamp. Because the *Alive* protocol is run almost continuously the reduction in messages can yield a significant performance benefit. The A*live* protocol is run after the structural hierarchy is in place so time synchronization should be easily achieved.

4 Conclusions

The analyses in this paper have shown that the Medusa protocol with some minor alterations can provide the security requirements: confidentiality, integrity and authentication, even with an active intruder out to subvert the Medusa process. Also, further alterations have been suggested to the various sub protocols to increase their efficiency whilst maintaining the security requirements. These changes 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 improvement. Furthermore if for some reason time synchronization in the latter phases is unavailable, the Medusa protocol can be adapted to avoid time synchronization altogether.

References

1. Birman, K. The Process Group Approach to Reliable Distributed Computing, In *Communications of the ACM*, 36(12), 1993, 37-53.
2. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D. The Role of Trust Management in Distributed Systems Security, *Secure Internet Programming*, J.Vitek, C. Jensen, ed., Springer-Verlag, 1999, 185-210.
3. Cachin, C. and J. Poritz (2002). Secure Intrusion Tolerant Replication on the Internet, *International Conference on Dependable Systems and Networks*, Washington.
4. Capkun, S., Buttyán, L., Hubaux, J.P. Self-Organized Public-Key Management for Mobile Ad Hoc Networks, *IEEE Transactions on Mobile Computing*, Vol. 2(1), 2003.
5. Daskapan, S. Dependable security by twisted secret sharing, *19th IFIP Information Security Conference*, Toulouse, 2004.
6. Daskapan, S., Verbraeck, A., Vree, W.G. The merge of computing paradigms, *5th Int.Conf. on computer and information technology*, Dhaka, 2002, 553-558.
7. Daskapan, S., Vree, W.G., Sol, H.G. Building a Distributed Security Defence System. In *Proc of the IEEE Int Conf. on Systems*, Man & Cybernetics, Delft, 2004.
8. Dolev, D., Yao, A.C. On the Security of Public Key Protocols, *IEEE Transactions on Information Theory*, 29(2), 1983.
9. Donovan, B., Norris, P., Lowe G., Analyzing a library of security protocols using casper and FDR. *In Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999
10. Ellison, R., Fisher, D., Linger, R., Lipson, H., Longstaff T., Mead, *N. Survivable network systems: An emerging discipline*, Tech. Report CMU/SEI-97-153, CMU, Pittsburgh, 1997.
11. Gong , L. (1993). "Increasing Availability and Security of an Authentication Service." IEEE Journal on Selected Areas in Communications 11(5): 657-662
12. Hoare, C.A.R. *Communicating Sequential Processes*, MIT Press, 1988.
13. Reiter, M. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart, In *Proceedings of 2nd ACM Conf. on Comp. and Comm. Security*, ACM, 1994, 68-80.
14. Renesse, R. van, Birman, K., Maffeis, S. Horus: A Flexible Group Communication system, In *Communications of the ACM*, 39(4), 1996, 76-83.
15. Roscoe, A.W. *The Theory and Practice of Concurrency*, Prentice-Hall, 1997.
16. Ryan, P. Y. A., Schneider, S. A. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison Wesley Publ. Co., Reading, Massachussetts, 2000.
17. Schneider, S.A., *Concurrent and Real Time Systems: the CSP Approach*,Addison-Wesley,1999.
18. Shamir, A., How to Share a Secret, *Communications of the ACM 22(11)*, 1979.
19. Wiechers, W.K., Daskapan, S., Vree, W.G. Simulating the Establishment of Trust Infrastructures in Multi-Agent Systems, In *6h Int. Conference on E-Commerce*, Delft 2004.
20. Zhou, L., Haas, Z. J. Securing Ad Hoc Networks, *IEEE Network Maga*zine, Vol. 13(6), 1999
21. Zimmermann, R. *The Official PGP User's Guide*, MIT Press, Cambridge, 1995.