A Chained Authentication Model for Virtual Organizations

Genge Bela[1], Haller Piroska[2],

***Abstract* — Digital certificates (through the use of public-key cryptography) provide an entity the power to authenticate users even when the CA (Certification Authority) that issued the certificate is not on line. This authentication mechanism is widely used today in Virtual Organizations (VOs) to distribute the weight of the authentication. However, as shown in this paper, the use of public-key cryptography may dramatically reduce the performance of a system, and consequently reduce the performance of the VO if it is used to authenticate users at every node. Thus, we outline the specifications of a Virtual Organization, where public-key cryptography is used only for the initial authentication of nodes. To provide access to multiple resources in the VO, a Chained Authentication Model is proposed that makes use of a symmetric-key based three-party authentication protocol.**

Keywords* — *Distributed Authentication, Protocol Formalization, Virtual Organizations.

I. INTRODUCTION

In today's Internet, the possibility of being the subject of an attack is growing each day. This is why companies restrict access to their stations by creating Virtual Private Networks (VPN) or by using proxies and secured gateways. However, these solutions are static and represent an authentication and communication bottleneck (for the protected networks).

This paper deals with a Mobile Virtual Organization type, where a node may move around, and may access any other node in the system, previously being authenticated by a third, trusted party. There have several proposals for the administration and authentication in Virtual Organizations ([1]-[2]-[3]-[4]-[13]), but the systems described consider a global access and authentication point, meaning that every node will have to consult a central authority when accepting a connection from a specific host and each client will need to have a password with each node it wishes to access.

These assumptions are perfectly reasonable if we consider systems that are static or protected by physical devices (routers, proxys) [4]. But today, the mobile world is emerging and VO's may accept each day new members who can offer new services.

Other solutions [20]-[21]-[22]-[23]-[24]-[25]-[26]-[27] use certificate hierarchies to distribute the weight of the authentication among Certification Authorities that do not have to be online for the authentication process to take place. Although these provide a real-life solution to the authentication problem in VOs, the performance of the entire Virtual Organization may dramatically drop if every user must be authenticated at each node using a time-consuming public key algorithm.

We propose an organization type having multiple points of access, where each node, modeled as an agent, can become a possible authentication point (i.e. *Authenticator*). To avoid affecting the performance of the Virtual Organization (VO), we use public-key cryptography only at the initial authentication point. For later authentications we use private-key cryptography and a third-party authentication protocol so that the system will be able to handle a large amount of users.

The proposed protocol is formalized and transformed into CSP specification [15] using the Casper compiler developed by Gavin Lowe [5].

The paper is structured as follows. Section II shows how symmetric and asymmetric encryption algorithms influence multimedia systems, thus, motivating the use of symmetric-key based authentication in the proposed VO. In section III we introduce our system, describing VOs in general, and then specifying the properties of the proposed Coordinated Mobile Virtual Organization. In section IV we describe the third-party protocol used in the process of authentication. We end with a conclusion and a specification of future work in section V.

II. THE NEED FOR DISTRIBUTED AUTHENTICATION

A centralized authentication model, where every node in the system must be able to access information about every user (through a database link, for example), may eventually become a bottleneck, leading to the collapse of the system under it's own weight. There have been

[1] Genge Bela is with the Faculty of Engineering, "Petru Maior" University of Targu Mures, Romania; bgenge@upm.ro
[2] dr. Haller Piroska is with the Faculty of Engineering, "Petru Maior" University of Targu Mures, Romania; phaller@upm.ro

several proposals made to distribute the weight of the authentication among system components [13]-[20]-[21]-[22]-[23]-[24]-[25]-[26]-[27], differentiated by the component that guarantees the authentication of a requesting user. Thus, we can identify authentication models that are based on *passwords* or *certificates*.

A. Password-based authentication

The *password* approach implies that there is a component of the system that stores the password of the user, used in the authentication process as a symmetric key for encrypting the communication channel between the user and the server. The most representative authentication model from this category is *Kerberos* [13], which uses a Key Distribution Center (KDC) to hold the user passwords. The KDC issues to clients short-lived credentials (a Ticket Granting Ticket - TGT), which must be presented to a Ticket Granting Service (TGS) from where clients obtain session tickets to access a certain server.

The major problem of a system that is based on a password-based authentication is that the password is used for authenticating by both the user and the server (considering that symmetric cryptography is used). Because of this, if someone is able to obtain a password, it can impersonate the user and the server as well.

B. Certificate-based authentication

Certificates are digital documents signed by a central Certification Authority (CA) in which all of the system components trust. A signature is created using an entity's private key and it can be verified by using the entity's public key. Thus, every component of a system that possesses a certificate also possesses a public/private key pair.

A certificate usually contains the name of the authority that emitted the certificate, the issue and expiration dates, the name of the entity for which the certificated is issued, the public key of the entity and possibly other information (which may be system dependent). All this is signed using the CA's private key.

When the certificate-based model is used, system components can prove their identity by simply signing information. They can verify a signature by using the entity's public key, which is retrieved in a certificate signed by the CA.

A major problem with using certificate-based authentication is based on performance, since public key (asymmetric) cryptography is known to be at least 1000 times slower than secret key (symmetric) cryptography. Another drawback is the maintenance of the certificate revocation lists, which must be stored on public servers that have to be regularly updated.

C. Performance evaluation

To construct a system that uses a certain authentication model, one must take care when evaluating the security requirements because security means lesser performance of the resulting system.

Also, it is important to know if the system should rely on a password or certificate based authentication because the problems that result for each implementation must be treated in a different manner.

The basic question that must be eventually answered is the following: how does symmetric and asymmetric cryptography affect the functionality of a system component? To answer this question, we have tested the effect of the RSA and AES algorithms on the normal functioning of a multimedia (audio and video) server. What we have tested actually was the performance flow of the server when running in parallel multiple authentication sequences that made use of the RSA or AES algorithms.

The test was made using an AMD Athlon XP 1700+ (1.47 GHz) with 256 MB of RAM desktop computer on which the server was running. We used 50 *User Threads* to create a load on the server, and several *Cryptographic Threads* that ran one encryption and one decryption of a 1024 bit buffer/second/thread (Figure 1) to simulate parallel authentication. The 1024 bit buffer simulates a 1024-bit key sent to a user.

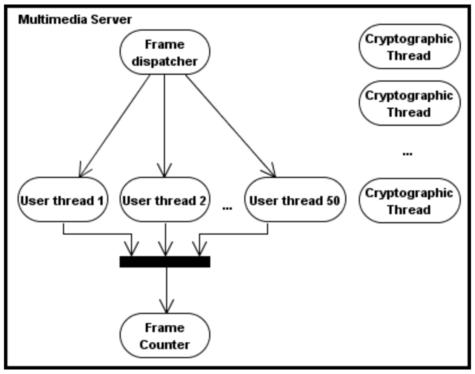

Figure 1. Multimedia test-server structure

The frames were dispatched to the *User Threads* by the *Frame Dispatcher* thread. A separate thread (*Frame Counter*) was used to count the number of frames that actually reached to each user thread.

Because the purpose of the test was to reveal the influence of one ore more authentication sequences on the functionality of an existing system, we did not use any encryption on the multimedia channels.

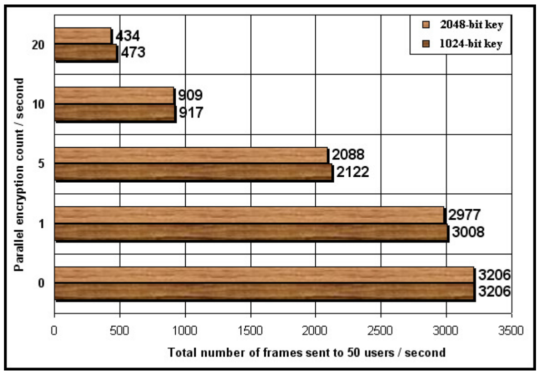

Figure 2. Asymmetric (RSA) encryption effect on the multimedia server

In Figure 2 we can see just how much does asymmetric cryptography cost. When running one encryption in parallel, the performance of the system drops from 3206 frames processed to 3008, for 1024-bit key, and to 2977, for a 2048-bit key. Thus, the system loses almost 10% only for one asymmetric encryption/decryption per second, which is not that critical, if we consider that there are 50 users logged in. However, a single authentication sequence may consist of several encryptions/decryptions [10], thus for one authentication, there would have to be run two or three encryptions+decryptions/second, resulting in 20-30% performance loss, which can become a serious concern for live streaming.

Figure 3 shows the effect of applying symmetric cryptography (AES algorithm) on the server. In this case, there is very little variation (±0.093) in the number of frames sent to users even when running 20 parallel encryptions.

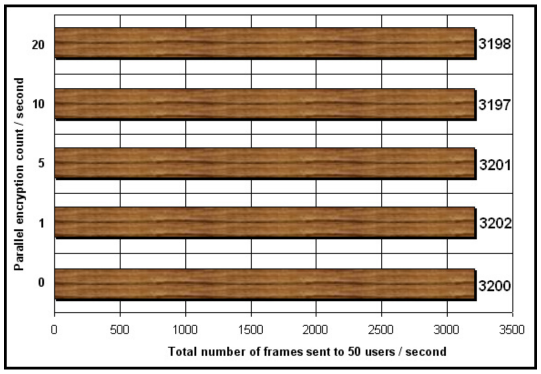

Figure 3. Symmetric (AES) encryption effect on the multimedia server

Knowing just how much a single encryption/decryption affects the performance of a server, we should consider using public key cryptography only when absolutely necessary. In addition, when working with multimedia servers, the authentication of a single user may affect the streaming of all other users, which in case of audio streaming is not permitted.

III. COORDINATED MOBILE VIRTUAL ORGANIZATIONS

A Virtual Organization (VO) is a set of entities (nodes), that we call *agents*, each of them having a set of resources that may be used by a specific client, or other agent. We consider nodes as being agents because they are not restricted to one point, they can move from one node to another to satisfy their goals. Examples of autonomous agent systems can be found at [1]-[6]-[7]-[8]. Because of the observations made in section II, the proposed VO uses an initial public-key based authentication and multiple subsequent private-key authentications.

The purpose of this paper is not to answer the question “why is a VO created?” or “how does it share resources?”. These questions are covered in detail in [1]. Instead, the questions that may find answers here, look like “is he safe to join the VO?”, “am I talking to the right person?” or “are you qualified to become an *Authenticator*?”.

A. System architecture

The system is composed of three kind of agents: *Coordinator* (*C*), *Authenticator* (*A*) and *Requestor* (*R*), as shown in Figure 4. These are connected through network lines that may be not permanent, or may be wireless connections, and more important, they are not safe: messages may be loosed, spoofed, replicated or created using old discovered passwords.

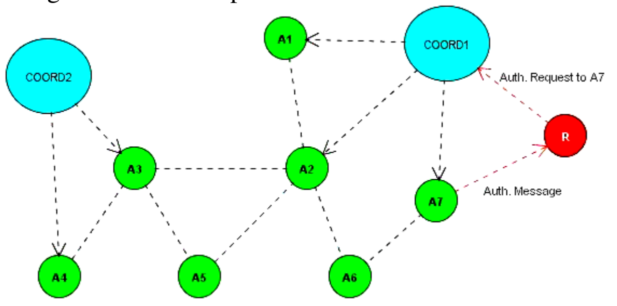

Figure 4. Agent R’s entry in the VO, using COORD1 as access point

A VO may have multiple *Coordinators* that may be connected to resource (*Authenticator*) nodes. Because connections to *Authenticator* nodes are made on request, when the system is started the *Coordinators* are not connected to any *Authenticators*.

The *Coordinators* play a crucial role in the initial authentication of nodes. For the initial authentication we have chosen the certificate approach because this allows an authentication even if the CA is not on-line and it may be done by any node that has the public certificate of the CA. The *Coordinators* may not be connected with all nodes in the VO, but they can authenticate users independently from each other by requesting the user's Certificate.

When a client (*Requestor agent* - *R*) wants to join the VO, to access the resources provided by an *Authenticator*, he takes his request to one of the *Coordinators*. Because this request is accompanied by a certificate, a certificate-based authentication protocol is used: SSL. If the user does not possess a certificate, a user password may be used to complete the protocol.

B. Registration

The process of registration can take any form, through a web page, e-mail, the important thing is that at the end, the user will have a certificate containing personal data (supported algorithms, password and random number generation capabilities, encryption, decryption speeds) signed by a Certification Authority (CA).

C. The agents

The ***Coordinator*** agent's role is mainly for primary authentication. We say mainly, because it will also have other purposes in the future, like the centralization of node behavior and key lifetime assertion. The system allows the access of two kinds of agents:

- *Authenticator*
- *Requestor*

If a new node wants to become an *Authenticator* ($A_i, i = N_A, N_A \geq 0$), he must first authenticate himself at one of the *Coordinators* ($C_j, 1 \leq j \leq N_C$), using his secret password $K_{A_iC_j}$ and presenting the certificate given by the CA. After the verification of the password and the certificate, C_j will engage in a provocation conversation with A_i, testing the "knowledge" of the new agent, so A_i can prove that it is capable of authenticating other users.

The conversation between the two parts consists of a sequence of question/response (X/Y) type messages as those described in the process of argumentation in Letia [8], the difference being that this conversation is based on challenging the opponent, existing only one proponent, the Coordinator:

- $Q'_i, i \geq 0$
- $R'_j, j = i + 1$
- $Q'_i \in X$
- $R'_j \in Y$

The proposed verifications include:

- Random number generation (rand)
- Supported algorithms (alg)
- Proposed message encryption speed (enc)
- Password generation (pwd)

Having these analyzed, a normalized quality function of the agent is constructed:

$$Qag(\ rand, alg, enc, pwd\ ) = (\,f(rand) + f(alg) + f(enc) + f(pwd)\,)\ /\ 4 \qquad (1)$$

where $f : R \rightarrow [0,1]$. The result is compared with the one in the database. The accepted tolerance function allowed, that is, the allowable difference between the value stored in the database at registration and the computed value is the following:

$$\eta = \left| Qag - D_Q \right| \leq \alpha \qquad (2)$$

where D_Q is the stored (registered) value of the quality of the agent, and α is the allowed tolerance level.

After the authentication process has been completed, the A_i agent is allowed to connect to the next *Authenticator* node, requesting the *Coordinator* to initiate the authentication algorithm described in the next section.

The ***Authenticator*** agent is responsible for the introduction of nodes wanting to migrate to other *Authenticators*. Because *Authenticators* stand for the actual resources in the VO, we consider the terms *Authenticator* and resource as having the same meaning. The requirements of the authenticated nodes are passed from *Authenticator* to *Authenticator*. These include a set of values $\{L_i, I_i, K_i\}$, where L_i is the lifetime of the new session key, I_i is the requirement of the new agent, to authenticate other agents, and K_i is the capability of the agent (password generation, algorithms, …).

The last agent is the ***Requestor*** agent, which may correspond to any entity that wants to request an authentication from the organization. If a newly authenticated entity has the possibility to authenticate other nodes, he will be an *Authenticator*. Else, he will remain in the state of *Requestor*.

This chain of authentication allows the system to be scalable and not to depend on the functionality of the *Coordinators*.

D. Chained-Authentication models

In this section we analyze how newly authenticated nodes can gain access to resources provided by the VO.

As stated in the previous section, the *Coordinator*'s role in the VO is to provide an initial authentication and *coordination* of new agents to the appropriate resources.

Newly authenticated agents send the list of resources they would like to access to the *Coordinator* (Figure 5).

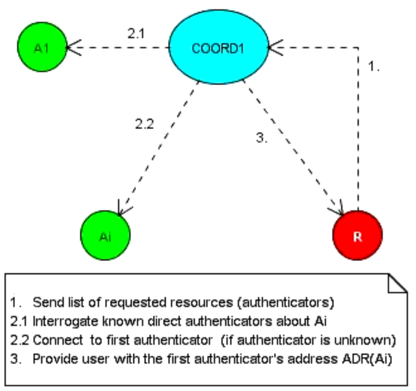
A1
COORD1
2.1
1.
2.2
3.
Ai
R
1. Send list of requested resources (authenticators)
2.1 Interrogate known direct authenticators about Ai
2.2 Connect to first authenticator (if authenticator is unknown)
3. Provide user with the first authenticator's address ADR(Ai)

Figure 5. Agent coordination process to the requested resources

Because resources may become unavailable, or highly loaded, it is the *Coordinator*'s role to find an available resource for the agent and return it's address.

The authentication process takes the form of a three-party authentication protocol where, in the initial state, the *Coordinator* plays the part of the third party.

After the coordination process is complete, nodes may move from one agent to another using the same three-party authentication protocol. This continuous authentication process is called a *chained-authentication*.

Next, we present three *chained-authentication* models, each of them having advantages and disadvantages.

D.1 Coordinator-based chained-authentication model

Because *Coordinators* share passwords with all the resources (through certificates), they have the ability to authenticate users at all the requested nodes, as shown in Figure 6.

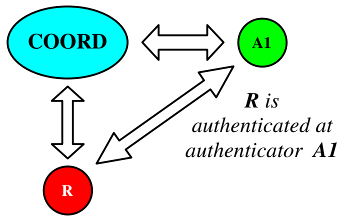

Figure 6. Coordinator-based authentication of agents at every resource (*Authenticator*) point using a third-party authentication protocol

Although this model solves the multiple resource access problem, if the VO has many users (it can vary from a couple of hundred users to ten thousands of users), the *Coordinators* can not handle the load, leading to the collapse of the entire VO.

D.2 Passive chained-authentication model

This model assumes that the *Requestor* agent has already been authenticated at a resource. This authentication model is *passive* from the *Requestor* point of view, because the actual authentication to the next requested resource is done by the current *Authenticator* node, which may imply multiple authentications, as shown in Figure 7.

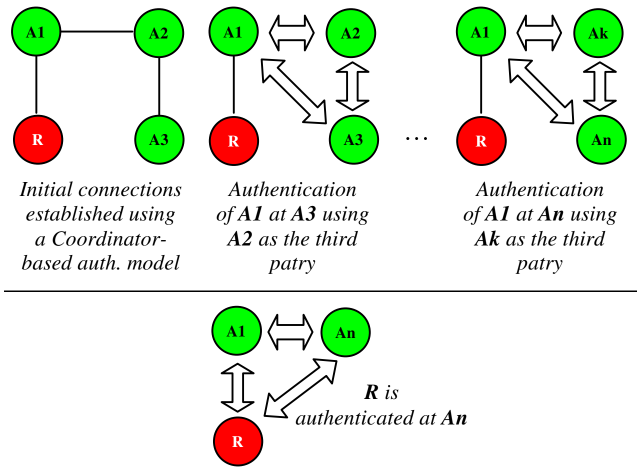
A1
A2
R
A3
A1
A2
R
A3
…
A1
Ak
R
An
Initial connections established using a Coordinator-based auth. model
*Authentication of **A1** at **A3** using **A2** as the third patry*
*Authentication of **A1** at **An** using **Ak** as the third patry*
A1
An
R
R** is authenticated at **An

Figure 7. Passive authentication of agents at next resource

Although this model removes the load from the *Coordinator*, it is up to the current *Authenticator* to find the next available resource, which may include multiple authentications. The user will stay *passive* until the *Authenticator* finds the next resource.

D.3 Active chained-authentication model

In this model (Figure 8), the current *Authenticator* authenticates the agent at the next resource, which may not be the next actual resource requested, but an intermediary node. This is why, in this model, the user is *Active*, constantly requesting authentication until it reaches the resource it wants to access.

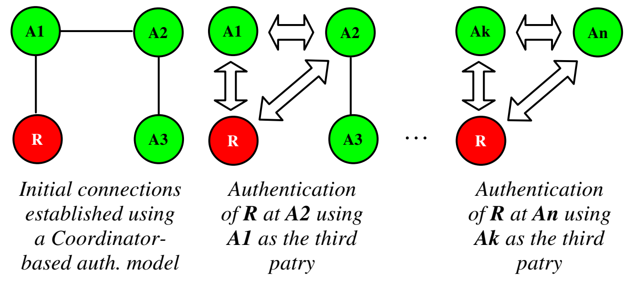
A1
A2
R
A3
A1
A2
R
A3
…
Ak
An
R
Initial connections established using a Coordinator-based auth. model
*Authentication of **R** at **A2** using **A1** as the third patry*
*Authentication of **R** at **An** using **Ak** as the third patry*

Figure 8. Active authentication of agents at every intermediary resource

Although the load of next authentication is removed from current *Authenticator*, it may be passed to an intermediary node or multiple intermediary nodes.

IV. AUTHENTICATION PROTOCOL

In the described system, there was a need for an authentication protocol that satisfied the following:

a) Support protocol runs in insecure environments
b) Support message loss
c) Support creation of session key by a capable third party
d) Minimize the contribution of random keys from the new entity
e) Use only symmetric algorithms
f) Minimize DoS and Replay attack possibility

To satisfy these needs, a number of existing protocols have been studied: "Wide-Mouth Frog" [9], Yahalom [9], Needham-Schroeder [10], Otway-Rees [11], Neumann-Stubblebine [12] and Kerberos version 5, as evaluated in [13] and all of them shortly presented in [14]. From these, only Kerberos satisfies almost all needs, the rest having the big problem that they are all open to DoS attacks because any user can initiate the authentication mechanism.

Kerberos could not be used in our system, because of the following. Firstly, it is too complex, relying on two authentication servers and timer synchronization, or in our distributed system, the only servers that need to communicate are adjacent and they may not have they're clocks synchronized. Secondly, because the key generation in Kerberos happens on every query for a TGT message, and if we consider a system where messages may be loosed, a new key will be generated even if the client did not get the last one. Also, a third party authentication protocol is more preferred because the "Man in the middle" attack may be harder to create if messages do not travel on the same line.

A. The Casper formalization language

This section briefly introduces the Casper security protocol specification language. For more information, the reader should consult [5].

The Casper project, developed by *Gavin Lowe* is composed of a specification language and a compiler. The Casper protocol specification language is simple and clear, making it possible to describe a protocol in a few minutes. The goal of the project was to offer a simple language, similar to the "usual" way of specifying protocols that would allow (using the compiler) transforming a protocol specification into a more complex, CSP description.

Next, we will briefly describe the syntax of the Casper language.

A Casper specification of a protocol is structured in sections. Because of space considerations we can not offer a full-description of each section, therefore we will focus our attention upon the following sections:

#Free variables
#Protocol description
#Actual variables
#System
#Intruder Information

In the '*#Free variables*' section, the user may specify the types of participants to the protocol (Agents, Servers), Key types (SessionKeys of ServerKeys), timestamp variables and so on. The naming of the variables chosen in this section is used in the '*#Protocol description*'.

The '*#Actual variables*' section contains the actual participants that take part to the run of the protocol. Using these variables, the System is specified in the '*#System*' section, stating the roles that each variable will take.

Finally, an intruder information is provided in the '*#Intruder Information*' section so that the protocol may be verified against the knowledge of the intruder.

The steps of a protocol are numbered according to the specification, larger messages may be broken into sub-steps by using letters:

2.a 2.b

Sending an encrypted message is possible using the following statement:

A -> B : { X, Y, Z }{ kab }

which means that A sends to B an encrypted message with the key 'kab' that is composed of 3 parts: X, Y and Z.

If a party does not need to understand a certain message, it must be specified with the special operator '%' meaning that the message is stored in a variable and sent to the destination principal later:

Store message into 'Va':

A -> B : { kab, Rs1, Ts }{ k } % Va

Send it to other principal:

B -> C : Va % { kab, Rs1, Ts }{ k }

B. The proposed protocol

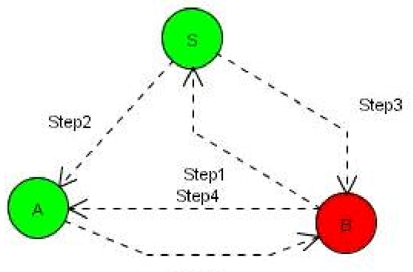

Figure 2. The algorithm steps for authenticating the new agent B

The '*#Free variables*' section:

A, B	*: Agent*
S	*: Server*
SKey	*: Agent -> ServerKey*
kab	*: SessionKey*
Ts, Tb	*: TimeStamp*
L, Rb, Rs, Rs1	*: Nonce*
InverseKeys	*: (SKey, SKey)*

The '*#Protocol description*' section:

```
0.      -> B : A
1.   B -> S : {A, Tb, Rb }{ SKey( B ) }
2a.  S -> A : { B, Rs, kab, L, Ts }{ SKey( A ) }
2b.  S -> A : { kab, Rs1, Ts }{ SKey( B ) } % Va
3a.  S -> B : { A, Rs1, kab, L, Ts }{ SKey( B ) }
3b.  S -> B : { Rb - 1 }{ kab }
3c.  S -> B : { kab, Rs, Ts }{ SKey( A ) } % Vb
4.   B -> A : { B, Vb % { kab, Rs, Ts }{ SKey( A ) } }{ kab }
5.  A -> B : { A, Va % { kab, Rs1, Ts }{ SKey( B ) } }{ kab }
```

The '*#Actual variables*' section:

Alice, Bob, Mallory	: Agent
Sam	: Server
Kab	: SessionKey
TS, TB	: TimeStamp
Life, RB, RS, RS1	: Nonce

The '*#System*' section:

```
INITIATOR(Bob, Sam, TB, RB)
RESPONDER(Alice)
SERVER(Sam, TS, Life, RS, RS1, Kab)
```

The '*#Intruder Information*' section:

```
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Sam,
                     SKey(Mallory)}
```

C. Protocol analysis

The protocol is straightforward, being initiated by agent Bob (the *Requestor*), who wants to authenticate himself to Alice (node A). S plays the role of the third-party server.

The protocol assumes the following:

i. A and S share a secret key SKey(A)
ii. B and S share a secret key SKey(B)

Although these keys are in fact session keys (established at the beginning, when the user authenticates himself to one of the *Coordinators*) we consider them server keys because of the roles they play.

To protect against replay attacks, the protocol makes use of timestamps. The clocks of communicating neighbors do not have to be synchronized because the timestamp is used only as a Nonce [16]-[17]-[18]-[19] ("Number once used"). This way, the receiving entities will not have to store a list of random nonces and verify them against incoming messages, but check only the timestamp of the latest package.

The protocol is started by B who wants to authenticate himself to A (step 0).

0. -> B : A

In step 1, B informs S that it wants to be authenticated to A:

1. B -> S : {A, Tb, Rb }{ SKey(B) }

B sends to S this message, composed of the name of A, a Timestamp Tb and a random number Rb, all encrypted with the key he shares with the server. The timestamp is sent to ensure S that this is a fresh message. The Rb is used to hide the contents of the message so that the protocol is well protected against offline-dictionary attacks.

Receiving this message, the server checks the timestamp and generates two messages, one of which is sent to A and the other one is sent to B. This way, A may present to B a proof that he is "known" by S and B may present to A a proof that he is also "known" by S. The word "known" is used to state that S has authenticated the parties.

The message sent back to A is decomposed in two parts for clarity. The first part:

2a. S -> A : { B, Rs, kab, L, Ts }{ SKey(A) }

informs A about the session key 'kab' that the server has generated. It also specifies a random number Rs that will be used by A to authenticate B. The package includes also a Lifetime for the key.

The second part:

2b. S -> A : { kab, Rs1, Ts }{ SKey(B) } % Va

is a message that A does not understand, beeing encrypted with B's server key. It is used only to prove that A knows the key and he got it from S. Also, in this package, the server ties the session key to the Rs1 random number so that the package may be authenticated with the random number sent back to B in step 3a.

The messages sent from B by S are composed of 3 parts. The first part:

3a. S -> B : { A, Rs1, kab, L, Ts }{ SKey(B) }

is a message similar to 2a, only addressed to B.

The second part:

3b. S -> B : { Rb - 1 }{ kab }

is used to ensure B that the server has produced the key and it is a response to the challenge sent by B in step 1.

The third part:

3c. S -> B : { kab, Rs, Ts }{ SKey(A) } % Vb

is similar to the message 2b.

After receiving these messages, the two parties now exchange the messages that will confirm them that the other side has received exactly the same password in the same run of the protocol. This is done in steps 4 and 5.

The great thing about this protocol is that it minimizes the use of B's capabilities in generating passwords and random numbers. The authentication process is grouped in sessions, the B agent having the capability of generating new authentication sessions if it wants to re-generate a password, or re-authenticate himself. If a message is loosed, B will not have to create a new authentication session, but he will only send a message for the same session, the keys being re-sent and not re-generated this way. The initial random number Rb is only used so that on response the client knows for which session was the authentication process started.

In the future, we will offer a formalization of the protocol using Typed-MSR where we will model and analyze the users and the possible intruders as specified in [16]-[17]-[18], and using the Typed SPI-Calculus [19] that will allow us the verification of the protocol.

V. CONCLUSION

The described system allows the distributed authentication of users that are previously registered at a *Coordinator*. The system does not differ very much from other authenticated virtual organizations when we are looking at the way entities join the system. The main difference is that users are not required to have a password with any of the entities in the VO and still be able to authenticate themselves properly using a third-party authentication protocol, as the one described in section III.

This system is recommended for use in mobile networks where agents (nodes) move around continuously, collecting information and then leaving the organization.

The protocol described in section III allows not only a third-party authentication, but was designed for the tolerance of message loss, and for use in environments that are not message-secure.

In this paper, we have presented a system that offers a decentralized authentication protocol where each node may become an *Authenticator*. As future work we plan to construct a simulation model for the system that will allow us to detect and correct the possible faults. Also, we will have to investigate on the possibilities of evaluating the security "fingerprint" (password and random number generation capabilities) of a specified user so that the password will only become a back-up security element and not the primary means of authentication.

In these kinds of systems, nodes may misbehave, may malfunction as the result of hardware/software error, generating random data in a Byzantine manner. This is why, we propose also as a future work, the introduction of behavior lists for each *Authenticator* agent so they may be excluded from the system in proper time.

REFERENCES

[1] Timothy J. Norman, Alun Preece, Stuart Chalmers, Nicholas R. Jennings, Michael Luck, Viet D. Dang, Thuc D. Nguyen, Vikas Deora, Jianhua Shao, W. Alex Gray, Nick J. Fiddian, "Agent-based formation of virtual organizations", at *KBS*, 2004.

[2] Roberto Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, Alberto Gianoli, Károly Lörentey, Fabio Spataro, "VOMS, an Authorization System for Virtual Organizations", *European Across Grids Conference*, 2003, pp. 33-40.

[3] Michael Kaminsky, George Savvides, David Mazière, M. Frans Kaashoek, "Decentralized user authentication in a global file system", at *SOSP*, 2003, pp. 60-73.

[4] Mark L. Green, Steven M. Gallo, Russ Miller, "Grid-Enabled Virtual Organization Based Dynamic Firewall", GRID 2004, Pittsburg, PA, USA, pp. 208-216.

[5] Gavin Lowe, "Casper: A compiler for the Analysis of Security Protocols", In *Proc. CSFW '97*, Rockport. IEEE, 1997.

[6] Ana L. C. Bazzan, "A distributed approach for coordination of traffic signal agents", AAMAS, 2005, 131-164.

[7] N. Haque, N. R. Jennings, L. Moreau, "Resource allocation in communication networks using market-based agents", KBS, 2005.

[8] Ioan Alfred Letia, "Gradually intrusive argumentative agents for diagnosis", Muti-Agent Systems for Medicine, Computational Biology and Bioinformatics, 2005.

[9] M. Burrows, M. Abadi, R. Needham, "A Logic of Authentication". *ACM Transactions on Computer Systems*, Feb 1990, pp. 18-36.

[10] R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", Communication for the ACM, Dec 1978, pp 993-999.

[11] D. Otway and O. Rees, "Efficient and Timely Mutual Authentication", Operating Systems Review, 1987, pp. 8-10.

[12] A. Kehne, J. Schonwalder, H. Langendorfer, "A Nonce-Based Protocol for Multiple Authentications", Operating Systems Review, Oct 1992, pp. 84-89.

[13] B.C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks", IEEE Communications Magazine, Sep 1994, pp 33-38.

[14] Bruce Schneier, "Applied Cryptography". John Wiley & Sons, 1996.

[15] C. A. R. Hoare, "Communicating Sequential Processes", Prentice Hall, April 1985.

[16] Balopoulos T. Gritzalis S. Katsikas S., "An Extension of Typed MSR for specifying Esoteric Protocols and their Dolev-Yao Intruder", in Proceedings of the CMS'2004 IFIP TC6/TC11 International Conference on Communications and Multimedia Security. D. Chadwick (Ed.), September 2004, Salford, UK, Kluwer Academic Publishers.

[17] C.J.F. Cremers, S. Mauw & E.P. de Vink, "Formal Methods for Security Protocols: Three Examples of the Black-Box Approach", NVTI Newsletter 7, 2003.

[18] Iliano Cervesato, "Typed Multiset Rewriting Specifications of Security Protocols", Electr. Notes Theor. Comput. Sci. 40, 2000.

[19] A. D. Gordon, A. S. A. Jeffrey, "Authenticity by Typing for Security Protocols", In J. Computer Security. 11 (4). 2003, pp. 451-521.

[20] Mary R. Thompson, Abdelilah Essiari, Srilekha Mudumbai, "Certificate-based Authorization Policy in a PKI Environment", *ACM Trans. Inf. Syst. Secur.*, 6(4), 2003, pp. 566–588.

[21] Laura Pearlman, Ian Foster, Von Welch, Carl Kesselman, Steven Tuecke, "A Community Authorization Service for Group Collaboration", in the Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks, Monterey, CA, USA, 2002, pp. 50-59.

[22] Markus Lorch, "PRIMA Privilege Management and Authorization in Grid Computing Environments", PhD Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, April 16th, 2004.

[23] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lorentey and F. Spataro, "From gridmap-file to VOMS: managing authorization in a Grid environment", *in Future Generation. Comput. Syst.* 21, 4, 2005, pp. 549-558.

[24] Marvin A. Sirbu, John Chung-I Chuang: "Distributed Authentication in Kerberos Using Public Key Cryptography", Symposium on Network and Distributed System Security, 1997, p. 134.

[25] J. R. Burruss, T. W. Fredian, M.R. Thompson, "ROAM: An Authorization Manager for Grids ", Journal of Grid Computing, 4(4), 2006, pp. 413-423.

[26] Rebekah Lepro, "Cardea: Dynamic Access Control in Distributed Systems", NAS Technical Report NAS-03-020, November 2003.

[27] Wolfgang Hommel: "An Architecture for Privacy-Aware Inter-domain Identity Management", In *Proceedings of the 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2005),* Barcelona, Spain, October 2005, Springer.