ROSA: Realistic Open Security Architecture for active networks[1]

Marcelo Bagnulo[1], Bernardo Alarcos[2], María Calderón[3], Marifeli Sedano[4]

[1,3] Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid.
Av. Universidad 30 - 28911 LEGANES (MADRID)
[2,4] Área de Ingeniería Telemática. Universidad de Alcalá
28871 Alcalá de Henares (MADRID)
{marcelo,maria}@it.uc3m.es, {bernardo,marifeli}@aut.uah.es

Abstract. Active network technology enables fast deployment of new network services tailored to the specific needs of end users, among other features. Nevertheless, security is still a main concern when considering the industrial adoption of this technology. In this article we describe an open security architecture for active network platforms that follow the discrete approach. The proposed solution provides all the required security features, and it also grants proper scalability of the overall system, by using a distributed key-generation algorithm. The performance of the proposal is validated with experimental data obtained from a prototype implementation of the solution.

1. Introduction.

Active networking technology[1] has already proven to be a powerful approach when fast deployment of new protocols and services is needed. However, security risks introduced by its own nature are a major concern when evaluating the usage of this technology in public environments. Furthermore, heavy security measures can preclude deployment in real scenarios because of the imposed overhead in terms of

[1] IWAN 2002, Lecture Notes in Computer Science 2546, pp 204-215. December 2002. This work has been funded by CICYT under project AURAS.

processing, bandwidth and/or latency. So, in order to achieve a deployable active network architecture, the security solution must not only provide protection from all the detected threats but it must also grant the scalability of the system. In this article, we will present ROSA, a Realistic Open Security Architecture for active network platforms that follow the discrete approach [2], which can fulfill both requirements thanks to a distributed key-generation algorithm and to architectural features of the discrete approach platforms.

The remainder of this article is structured as follows. In section 2 an introduction to discrete approach to active networks is presented. In section 3, the security solution requirements are detailed, including threats assessment and scalability requirements. Next, section 4 provides an overall description of the proposed security architecture. In section 5, implementation is described and performance results are discussed. Section 6 is dedicated to related works. Finally, section 7 is devoted to conclusions.

2. Active networks: the discrete approach

There are two different approaches to provide dynamic network programmability to active networks. Some active network platforms follow a discrete approach. This means, packets do not include the code to be executed in the Active Routers, but a separate mechanism exists for injecting programs into an Active Router, such as a *Code Server*. Other active network platforms follow an integrated approach, and packets (called capsules) include not only user data but the code used for the forwarding of the packet as well. We will next present three active networks that follow the discrete approach: DAN, SARA and ASP. These platforms are compatible with ROSA. Finally, we will describe the active packet exchange.

2.1. Discrete Approach platforms

DAN[3], which stands for Distributed Code Caching for active networks, has been developed by the Washington University of St Louis and by the Computer Engineering and Network Laboratory of Zurich. DAN is an Execution Environment (EE) that is running in the high performance Active Network Node, ANN [4]. The proposed framework mainly includes the following components: an Active Module Loader which loads the active modules authenticated and digitally signed by their developers from well known code servers using a lightweight network protocol (e.g. UDP/IP); a Policy Controller which maintains a table of policy rules set up by an administrator, e.g. restrict the set of supported modules; a Security Gateway which allows/denies active modules based on their origin and developer by analyzing their digital signatures/authentication information; a Function Dispatcher which identifies references to active modules in data packets and passes these packets to their corresponding function implementations; and a Resource Controller for fair CPU time sharing among active functions.

The next active network platform considered is SARA (Simple Active Router Assistant) [5] which is an active router prototype developed by the University Carlos III de Madrid in the context of the IST project GCAP [6]. It is based on the router-assistant paradigm, meaning that active code does not run directly on the router processor but on a different device, called assistant, which is directly attached to the router through a high-speed LAN. Hence, the router only has to identify and divert active packets to its assistant. Active packets are identified by the router alert option, enabling active router location transparency, since active packets need not be addressed to the active router in order to be processed by it. After requested processing is performed by the assistant, the packets are returned to the router in order to be forwarded. The active code needed to process active packets is dynamically downloaded from Code Servers when it is not locally available in the assistant. In this way, safety can be checked in advance, since only registered code proved harmless is stored in code servers. SARA is available in two platforms: One fully based on linux [7] (playing both roles: router and assistant as a development scenario) and a hybrid platform where the router used is an Ericsson-Telebit AXI462 running a kernel adapted to work with an active assistant.

Finally, we will consider one of the principal EEs running within ABone [8] which is ASP [9] from the University of the Southern California/Information Sciences Institute. In the implementation proposed in this EE, the active code is downloaded from a set of known secure code servers. An important contribution of ASP is the support of persistent active applications that may have long-lived execution threads.

2.2. Discrete platform packet exchange

In order to present the security architecture, we will first introduce the packet exchange performed, so we can detect the requirements imposed by security and scalability concerns.

The elements involved in the packet exchange are:

Source: User terminal that generates traffic and uses the active services.

Destination: It is the terminal that *Source* addresses its traffic to.

Active Router: It is a router capable of processing active packets. It is also able to obtain the active code needed.

Code Server: It is the active code repository that serves the *Active Routers*.

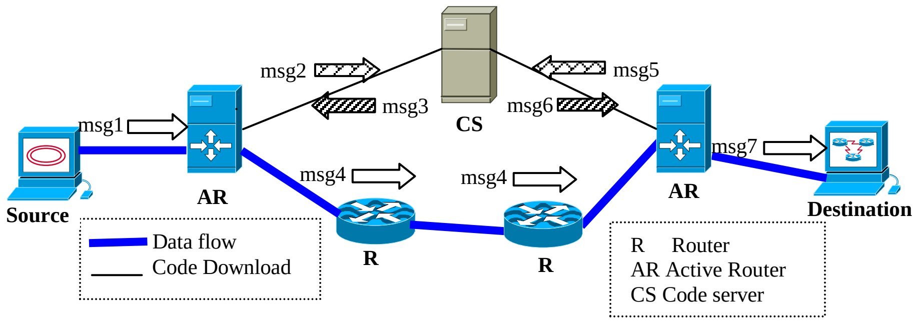

Figure 1. Services Network Architecture.

The packet exchange description depicted in figure 1 is described next. When *Source* needs special active processing for a flow of packets between itself and *Destination*, it must send packets (msg1), addressed to *Destination*, containing the identification of the active code that it desires to be executed. When a packet reaches the first *Active Router*, it is inspected and the identification of the active code is extracted. If the active code is locally available at the *Active Router*, it performs the requested process and then forwards the packet (msg4). If the needed active code is not locally available the *Active Router* requests it from the *Code Server* (msg2). The *Code Server* then sends the requested code to the *Active Router* (msg3), which now processes the packet and forwards it to the next hop. The same procedure is executed by all the *Active Routers* along the path (msg5, msg6), until the packet reaches *Destination*, where the packet is received (msg7). Next active packets of this flow will presumably follow the same path, so the *Active Routers* will be capable of processing them without needing to request the code from *Code Servers* again.

3. Security Architecture requirements.

In this section we will present the different requirements imposed on the security architecture. We will first start by stating the security requirements and then we will describe other general requirements, specially emphasizing the scalability aspects.

3.1. Security Requirements

Security requirements imposed by active networks have already been detailed in several documents [10]. So, we will not perform an exhaustive analysis here, but we will only present the final security requirements from each element´s perspective.

From the *Active Router´s* perspective, authorization is a key requirement. It is relevant that the active code loaded into the routers is provided by an authorized *Code Server* and not from an unauthorized source. In addition, the code integrity must be preserved while it is transmitted from the *Code Server* to the *Active Router*. In addition, the *Active Router* must be able to verify that the user that is requesting the code (*i.e. Source*) is authorized to execute it at this moment.

From the *Code Server´s* perspective, it must be able to authen ticate *Active Routers* that are requesting active code, since not all the code will be available to all routers. Furthermore, the security solution must provide confidential code transfer, in order to prevent unauthorized parties from inspecting the delivered code.

From the *Source´s* perspective, it must be able to be certain no other user is requesting active services on its behalf. It must also be the only one capable of controlling its active services, meaning that no other user is capable of introducing new active packets or modifying active packets sent by *Source*, interfering with the requested active service. In addition to authentication features, it is also important to provide non-repudiation; this is specially important when active services will be provided in a commercial fashion.

From the *Destination´s* perspective, there are no requirements since it does not demand active services from the network. It should be noted that end-to-end security is out of the scope of this security solution.

3.2 Other general requirements

Besides security requirements, the security architecture must also meet scalability and performance requirements, which are reflected next:

1. Zero user knowledge at the *Active Routers:* In order to build a manageable solution, user management must not be performed on each and every *Active Router*. A database containing all the users information, including access rights would be the preferred solution.
2. Path transparency: It must not be required that the *Source* be aware of which *Active Routers* are in the path used to transport packet towards the *Destination*. In addition, it must not be required that each node in the path has knowledge of its active neighbors. These requirements are needed to grant the scalability, performance and flexibility of the active network, since, hop-by-hop authentication is considered to be incapable of providing the mentioned features.

4. ROSA Security Architecture

In this section we will present the proposed security architecture. We will first consider S*ource* authorization issues, evaluating the different authorization paradigms available and then infering which one is the most appropriate for this particular problem. Then we will consider the code downloading security and non-repudiation issues. Finally, we will present the overall solution step by step.

4.1 *Source* Authorization

4.1.1. Authorization paradigm

A key feature that must be provided by the security architecture is authorization, i.e. *Sources* must be authorized to execute the solicited code on *Active Routers*. There are two authorization paradigms that can be used: authorization based on access control lists or authorization based on credentials. The first paradigm is based on the existence of an access control list (locally available or in a remote location) that must be queried every time an *Active Router* receives an active packet sent by *Source*, in order to validate the *Source´*s permissions. In this case the identity of the requesting party must be authenticated in order to prevent impersonation. This approach then requires that the requested device (*Active Router)* has information about *Sources* and permissions, or it imposes a communication with an authorization server every time a *Source* sends an active packet. The second paradigm demands that every time *Source* sends a packet, a credential that proves the *Source´*s permissions must be presented. Then the requested device (*Active Router*) only needs to verify the credential. However, credential generation and distribution may be more than a trivial task.

The solution proposed in this paper will be designed based on the second paradigm, since we consider that it provides better scalability attributes.

4.1.2. Considering the usage of public key cryptography.

In order to allow the intended use, a credential must contain verifiable authorization information, i.e. the permissions granted to the holder of the credential. In addition, it must be possible to verify that the issuer of the credential is a Valid Issuer, i.e. that it has the authority to grant these permissions. It is also critical to validate that the user that is presenting the credential is the same user that the credential was granted to.

In order to fulfill the above stated characteristics of a credential, public key encryption can be used. So, a credential containing the *Source´*s permission and the *Source´*s public key is signed by the Valid Issuer. Then, the *Active Router* must be capable of verifying the authenticity of the credential and also it must be capable of verifying that the requesting user has the private key that corresponds to the public key included in the credential. This mechanism provides all the required features, but the usage of public key cryptography is very demanding in terms of processing.

4.1.3. Authorization and key generation solution

In order to obtain a less demanding solution, symmetric key cryptography can be used. However, building a similar system using symmetric key would require the usage of two different symmetric keys (a first one shared by the Valid Issuer and the *Active Routers* and another key shared by *Source* and the *Active Routers*). This system would still demand two cryptographic verifications and it would present the additional problem of key distribution. So, in order to improve the scalability of the solution, we will next explore the possibility of using only one symmetric key, shared by the Valid Issuer, *Source* and the *Active Routers.*

The requirements imposed on this key are:

– Different keys for different *Sources*. (i.e. the key must be linked to a *Source*)

– Different keys for the same *Source* at different moments (i.e. the key must have a validity period)

- Different keys for different active codes requested by the same *Source* (i.e. the key must be linked to an active code/active service)

Therefore, the key K issued by the Valid Issuer is linked to a *Source*, an active code and a validity period.

Then, if K is used for generating an HMAC [11] included in active packets that requests the execution of a particular active code, the active packets themselves play the role of credentials. Basically, an *Active Router* receives an active packet that includes the requested code identification, the *Source* identity, the time when the active service was requested, the requested period and an HMAC. Then, if the *Active Router* has a valid key K linked to the *Source*, the requested code identification and the validity period, it can verify the authenticity of the active packet, without any further information. This mechanism imposes the usage of an A*uthorization Server* (the Valid Issuer role), that generates the keys (K). So, in order to execute a code in the network, *Source* must obtain the correspondent key (K) from the A*uthorization Server* in a secure way. This is not a time critical task, since it is only performed when the service is requested and it is possible to be executed in advance. However, once the service is authorized and the key K is generated, the *Authorization Server* must communicate it to all *Active Routers* in the network, so they are aware of the new authorization. This does not seems to be the most scalable solution, because of the amount of communications needed between the *Authorization Server* and the *Active Routers*.

We will next present an improved solution that minimizes the required interactions between these elements. The basic idea is that the key, K, can be almost autonomously generated in every *Active Router* when it is needed. In order to achieve this, we will associate a key (Kci) to every active code (Ci) that can be loaded in the *Active Routers*. These keys, Kci, are known by the *Code Server* and by the *Authorization Server*. Then, when a *Source*, S, requests authorization for the execution of code Ci at a moment T and for a period P, the *Authorization Server* generates the secret key K as the HMAC of the concatenation of the parameters Kci, S, T, P and Ci.

$$K = \mathrm{HMAC}(Kci, S, T, P, Ci)$$

The key K is then transmitted to the *Source*, so it can generate the HMAC that will be included in active packets with it. If we analyze the characteristics of K we can see that: K is linked to an active code (Ci, Kci); K is linked to a *Source* (S); K has a validity period (T, T+P); K can not be generated by the *Source,* since it does not have Kci. In addition, the *Code Server* can attach Kci to the active code when this is confidentially downloaded to the *Active Routers*. So, the *Active Routers* are capable of regenerating K without contacting the *Authorization Server* every time an active packet arrives or when a new *Source* requests an already downloaded code. The *Active Routers* have all the information needed to generate K, i.e. S, T, P and Ci are included in all active packets and Kci is obtained when they download the code from de *Code Server*.

Note that since the solution is based on shared secret keys, the security level of the solution can be defined by setting the number of parties that share the Kci keys (authorized *Active Routers* for a given code Ci) and the frequency with which Kci keys are changed. A re-keying procedure based on the usage of multiple overlapped keys has been defined, which can be easily tuned to obtain the requested security level. It should also be noted that, since Kci are stored in routers, it is assumed that routers have some form of secure storage capabilities.

4.2. Code downloading.

Another key feature that must also be provided is a secure way to download code (and Kci keys) from the *Code Server* into the *Active Routers*. However, this is not as time critical as user authorization since it is only performed once, when the first packet arrives to an Active Router. The subsequent packets will benefit from a cached copy of the code and the Kci. So, a protocol that allows a secure communication between two parties is needed. We will use TLS [12] since it provides all the needed features. Then both *Code Server* and *Active router* must have a digital certificate (public key cryptography is used), and a TLS session is established between the *Code Server* and the *Active Router*, before the code is downloaded.

4.3.- Non-repudiation.

When the user is requesting a service, commercial and legal issues may be involved, so non-repudiation is relevant. Furthermore, the security architecture can be used to enable charging mechanisms. In this case non-repudiation is considered as an important asset. In order to assure non-repudiation, public key cryptography must be used when the user requests authorization to the *Authorization Server*, as will be described in the following section.

4.4.- The security solution: step by step.

In this section we will describe the complete mechanism, illustrated in figure 2. First (step 1 in figure 2), the *Source* requests authorization (to the *Authorization Server*) to execute an active code Ci in the network (getting an active service). This request is done in a secure way, meaning that public key cryptography and digital certificates are used by both parties. Therefore, *Source*´s request is signed with the private key of *Source* and its digital certificate is also included. This request is encrypted with the public key of the *Authorization Server*. Then the *Authorization Server* after receiving and verifying the request, it generates K as the HMAC of the *Source*´s identification (S), the requested code's identification (Ci), the key associated to this code (Kci), the service request time (T) and the validity period requested by *Source* (P), as we presented in section 4.1.3. Then, the *Authorization Server* sends a signed message containing K. The message is encrypted with the public key of *Source*.

The *Source* decrypts the message and obtains K. Then (step 2 in figure 2), it generates active packets, that include its own identification S, the service request time T, the validity period P and an identifier of the requested active code Ci. This message includes an HMAC of the message using K.

$$\text{ActivePacket} = (\text{Ci}, \text{S}, \text{T}, \text{P}, \text{Payload}, \text{hmac}_K[\text{Ci}, \text{S}, \text{T}, \text{P}, \text{Payload}])$$

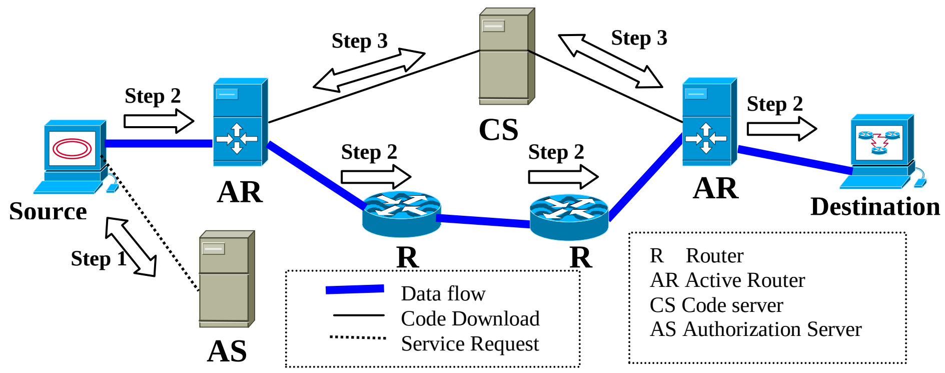

Figure 2. The diferent steps.

When an *Active Router* receives the message, it first verifies that the message is not obsolete, i.e. it is within the validity period, and then it verifies the solicited active code availability. In case the code (and Kci) is not locally available, it downloads it, using a secure (TLS) connection from the *Code Server* (step 3 in figure 2). Then the *Active Router* generates K, using Ci, S, T and P extracted from the active packet and Kci obtained from the *Code Server* when the code was downloaded. If the HMAC is verified, it means that *Source* has been authorized to execute the requested code, so the *Active Router* processes the packet using the requested code and forwards it to the next hop. The same procedure is repeated on every *Active Router* along the path until the packet reaches *Destination*. The subsequent active packets of the flow will benefit from cached copies of the active code and Kci in every *Active Router*. It must be noted that the presented solution is limited to one security domain, i.e. one *Authorization Server* providing keys. It is possible to extend the solution to multiple domains, but this is more than a trivial task and it will be presented in future works.

5. Implementation and tests.

In order to evaluate the viability of ROSA, the security architecture has been implemented and its influence on the end-to-end delay has been measured. Two main processes have been evaluated: active packets protection and secure active code downloads. We have not considered the service request phase, because it has a similar cost to a code download and it is only executed once.

In order to enable a simple integration of the developed prototype with available active network platforms, this implementation has been developed in Java, even though we are fully aware of the performance penalty of this choice.

5.1. Active packets protection cost.

In ROSA, active packets protection is provided by HMAC, so performance of HMAC Java implementation has been measured. Tests have been done using a PIII 1.1Ghz, 256MB, Linux Kernel 2.4.17 and JSKD 1.4.0. We have measured the time needed for performing an HMAC and its verification. Two different algorithms were considered, MD5 and SHA1, and the data block size ranged from 0 to 64KB. The results are that HMAC delay is between 0.38 ms and 4.55 ms.

5.2. Secure code downloads cost.

The test-bed used for this set of trials is as follows: a PIII-600 MHz, 64MB has been used as *Active Router* and a PIII-1.1GHz, 256MB has been used as *Code Server;* both systems have been directly connected via a Fast-Ethernet. The *Code Server* is a web server Apache v.1.3.24 with the SSL module. The code has been downloaded opening a TLS connection inside a previously established TLS session between the *Code Server* and the *Active Router*. The delay of the code download has been measured for non-secure connections (http) and secure connections (https). Different code sizes (from 1KB to 16 KB) have been used in the tests. The obtained delay is 3-5ms for http and 32-37 ms for https. In our scenario, the *Code Server* and the *Active Router* will be in the same domain so we estimate an additional delay of 10ms, which would be the mean delay introduced by 3 hops. Hence, for instance the estimated delay for 2KB code size would be 14 ms for http and 42 ms for https.

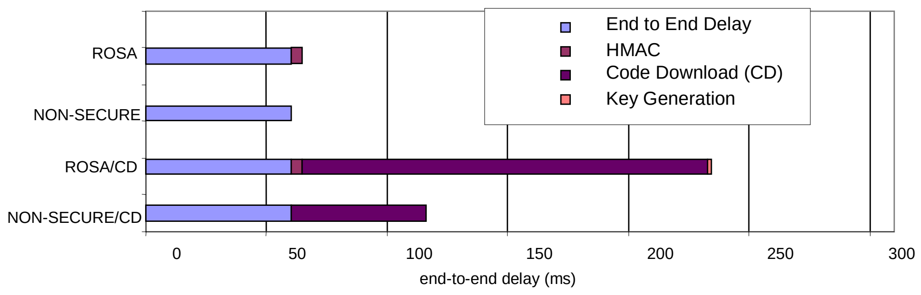

Figure 3. Comparison of end-to-end delay.

5.3 Security cost of an end-to-end typical communication.

In this section, we will evaluate the cost of providing security to the active network inside a typical Internet scenario, composed of 15 routers, four of which are *Active Routers*, with an average end-to-end delay of 60ms[2]. We will state the following additional suppositions: average packet size 512B; average active code size 2KB and all *Active Routers* modify active packets (so they must compute HMAC twice, one time to verify the received packet and another time to send the packet). Obviously, the active packet modification delay will be the same for the secure or non-secure solution, and therefore it will not be considered in the end-to-end delay.

In a non-secure scenario (NON-SECURE/CD in figure 3), the first active packet suffers a delay of 116ms (60ms end-to-end delay plus 4*14ms for code download). Note that the final CD indicates that this packet has led each *Active Router* to do a code download. In the ROSA scenario with code download (ROSA/CD in figure 3), delay increases to 234.4ms because of: https code download (4*42ms), key generation in *Active Routers* (4*0.46ms), HMAC in *Active Route*rs, *Source* and *Destination* (10*0.46ms).

[2] Data obtained from http://www.caida.org.

Next, we will analyze the cost when the active code is already in the *Active Routers*, which will be the situation in most cases . The delay introduced by ROSA is of 4.6ms (from 60ms in NON-SECURE solution to 64.6ms) because of the HMAC processed in *Active Routers*, *Source* and *Destination*.

The obtained results show that ROSA introduces a small increase to the non-secure end-to-end delay (7.6%) in most cases, that is, when code downloads are not needed. And, only the first active packet of the session experiences a higher delay. So, we conclude that the delay cost introduced by ROSA is reasonable and the proposed architecture is feasible.

6. Related work

In this section, we will perform a comparative analysis of the proposed solution, ROSA, and other security architectures proposed for other platforms. In particular, we will study the security solutions presented for ANTS, DAN and SANE.

ANTS [13] and ASP are the two main EEs running on Abone. ASP specification does not define a security architecture but SANTS [15] is an ANSA based proposal for ANTS. ANSA is an Active Network Security Architecture (ANSA) [14] proposed by the research community to be used in Abone [16]. ANSA uses symmetric key techniques (i.e. HMAC) over the variable part of the packet [17], in order to provide inter-node protection, and digital signature over the fixed part of the packet, to provide authentication and authorization of the principal. In ROSA we avoid asymmetric key with the purpose of improving the performance. That is possible because we do not need the non-repudiation service over the active packets flow, since we provide it during service request. Furthermore, since a topology independent solution is required, the mechanism to share the keys based on the neighboring relationship proposed in the ANSA based EEs [18] is not acceptable. Then, ROSA proposes a more sophisticated mechanism to distribute the key between the trusted components of SARA.

In DAN the security issues are addressed through policy and cryptography. The security problem is reduced to the implementation of a simple policy rule on the node which lets it choose the right code server and a database of public keys to check the developer’s signature of the plug-in and the code server’s authentication. DAN simply does not address additional security issues considered in the design of ROSA.

SANE [19] is a layered architecture developed in the University of Pennsylvania. The lower layer of the architecture use a secure bootstrap mechanism called AEGIS [20] that ensures that the system starts in an expected and safe state. SANE allows users to run their own modules on active nodes. In order to ensure the proper usage of network resources, it authenticates and authorizes requesting users through the usage of a modified version the Station To Station protocol (STS) [21] between the user and each active node along the path that packets will follow. Once the STS protocol has concluded, and a security association is established between the user and each active node, so that user´s packets can be authenticated. This scheme imposes the usage of a different authenticator for each active node in the path, which must be carried in each active packet. The proposed solution for this issue is that a common secret key is distributed among every active node using the established security association. The main drawback detected is the time needed for path establishment, since a STS exchange and a secret key exchange are needed. Moreover, when the path changes, these operations must be performed over the new path. ROSA has a reduced path establishment time since only one key exchange (with the *Code Server*) is needed in the worst case.

7.- Conclusions

We have presented a security solution for active network platforms that follow the discrete approach. Key features of the solution include: The solution performance is guaranteed by the usage of symmetric key cryptography. The scalability of the solution is assured by the authorization model, based on credentials, and the key distribution mechanism, that minimizes key exchanges by allowing key generation at every *Active Router* in an autonomous fashion. The security level of the solution is determined by the re-keying frequency i.e. how often Kci keys are changed. Essential features of the solution such as performance and scalability have been validated with measures obtained from a prototype implementation of the solution. Furthermore, it must be stressed that the proposed architecture is open since it is valid for any active network platform as long it follows the discrete approach using a *Code Server.*

References.

1 Wetherall, D. J., Legedza, U., Guttag, J.: Introducing new Internet services: Why and How. IEEE Network Magazine, 1998.

2 Tennenhouse, D. L., Wetherall, D. J.: Towards an Active Network Architecture Computer Communication Review. Vol. 26, No. 2, April 1996.

3 Decasper, D., Plattner, B.: DAN: Distributed Code Caching for Active Networks. IEEE Infocom'98. San Francisco, California, March/April 1998.

4 Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B.: A Scalable, High Performance Active Network Node. IEEE Network, Jan 1999. Vol.13, num.1, pag 8-19.

5 Larrabeiti, D., Calderón, M., Azcorra, A., Urueña, M.: A practical approach to network-

based processing. 4th International Workshop on Active Middleware Services, July 2002.

6 GCAP IST project home page. http://www.laas.fr/GCAP

7 SARA home site. http://enjambre.it.uc3m.es/~sara.

8 Berson, S., Braden, B., Ricciulli, L.: Introduction to the Abone. February 11, 2002. URL: http://www.isi.edu/abone/DOCUMENTS/ABarch/

9 Braden, B., Cerpa, A., Faber, T., Lindell, B., Pillips, G., Kann, J., Shenoy, V.: Introduction to the ASP Execution Environment (v1.5). November 30, 2001.

10 IST-FAIN Project, Deliverable “Initial Active Network and Active Node Architecture” Editor Spyros Denazis, 2001.

11 Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104, April 1997.

12 Dierks, T., Allen, C.: The TLS protocol Version 1.0. RFC2246. January 1999.

13 Wetherall, D., Guttag, J., Tennenhouse D. L.: ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. Proceedings IEEE OPENARCH98, April 1998.

14 AN Security Working Group. Security Architecture for Active Nets. November 13, 2001.

15 NAI Labs ANETS Group, SANTS Security Overview (May 18, 2000), URL:ftp://ftp.tislabs.com/pub/activenets/SANTSsecurityoverview.doc.

16 Faber, T., Braden, B., Lindell, B., Berson, S., Bhaskar, K.: Active Network Security for the ABone. November 30, 2001.

17 Lindell, B.: Protocol Specification for Hop-By-Hop Message Authentication and Integrity. Dec. 1999.

18 Murphy, S., Lewis, E., Puga, R., Watson, R., Yee, R.: Strong Security for Active Networks. Proceedings IEEE OPENARCH01. April, 27 2001.

19 Scott Alexander, D., Arbaugh, W., Keromytis, A., Smith, J. A Secure Active network architecture: Realization in the SwitchWare. IEEE Network, special issue on Active and Programmable Networks, May/June 1998, vol 12, no. 3, pp. 37-45.

20 Arbaugh, W. et al. Automated Recovery in a Secure Bootstrap Process. Network and Distributed Systems Symposium, Internet Society, March 1998.

21 Diffie, W., van Oorschot, P., Wiener, M.: Authentication and Authenticated Key Exchanges. Design, Codes and Cryptography, vol. 2, 1992.