# Performance of a Jini-based Ad Hoc Network Authentication Scheme

**M. M. McMahon, D. M. Needham, J. B. Datko**
**Department of Computer Science**
**United States Naval Academy**
**Annapolis, MD 21402-5002**

**Abstract**

Ad hoc networks provide a dynamically reconfigurable network infrastructure where entities can participate at will. Jini technology can be used to implement an ad hoc network, and to permit a user to enter the network and become a client of services without *apriori* knowledge of either the existence of the network or its services. However, Jini lacks security provisions for authenticating users of the network. Traditional Public key Infrastructure (PKI) approaches to authentication in an ad hoc network are static and cumbersome. An alternative approach that employs PKI in a dynamic manner, consistent with the rapidly changing nature of ad hoc networks, can provide the basis for such ad hoc authentication.

This paper presents an implementation of an algorithm that provides a foundation for secure, ad hoc joining of a distributed, mobile, wireless network. The approach is based on incorporating the use of a dynamic PKI approach with the Jini architecture. We describe the Java-based implementation of a certification authority hierarchy that supports our authentication process, and analyze the performance of our implementation. We also examine the influence of the Remote Method Invocation Daemon (RMID) in the performance of our implementation.

**Keywords**: Distributed Computing, Ad Hoc Networking, Ad Hoc Security, Mobile Network Security, Jini architecture.

## 1 INTRODUCTION

Although a Jini-based architecture can be used to implement a distributed ad hoc network, it lacks provisions for security when entities join the network. Security in such a networking environment is made complex by the rapid mobility, inherent uncertainty, and physical size/power constraints of the ad hoc networking entities. Further, Perkins [11] has shown that trusted third party authentication schemes have not incorporated well into mobile environments.

A conventional Public Key Infrastructure (PKI) is cumbersome for ad hoc connecting entities to use because most protocols involve frequent trusted-third party access. Security practices that include trusted third party referral, and require the ability to perform high computation public-key actions, do not function properly in the ad hoc, wireless domain. Most mobile entities are power-constrained, and therefore have limited computational power. Additionally, entities may not be close enough, in either a physical proximity or network sense, to access a trusted third party. Physical security is more easily controlled in wired networks; wireless networks are more susceptible to eavesdropping, denial of service attacks, and other malicious network behavior. Establishing secure communications in a dynamic environment is further challenged by the lack of *apriori* knowledge of other network entities [11].

We propose a hybrid encryption scheme for entities communicating on an ad hoc Jini network, built upon both symmetric (Advanced Encryption Standard (AES)) and asymmetric (RSA) encryption techniques. Entities communicate by exchanging a one-time symmetric session key. This symmetric key is distributed by encrypting it using a recipient's public key. Our authentication algorithm verifies the identity of an entity's public key using a certificate chain. When authentication ends successfully, a session key is sent to the joining entity. This session key allows the joining entity to participate in the network by requesting services, or registering its own services.

This paper presents an implementation of an algorithm that allows an entity to join a distributed, ad hoc network in a secure fashion. The remainder of this paper is organized as follows. Section 2 presents background about Jini ad hoc networking and certification authorities. Section 3 discusses the requirements and specifications for a distributed, ad hoc joining algorithm. In Section 4, we present the authentication experiments and an analysis of those experiments. In Section 5, we present our analysis. Section 6 contains our conclusions. Section 7 discusses our ongoing and future efforts.

## 2 BACKGROUND

This section describes the Jini architecture, certification authorities, and discusses related work in the area of mobile ad hoc network security.

### 2.1 Jini Architecture

Current distributed architectures do not provide sufficient support for dynamic, distributed, ad hoc networks, and are inherently limited by being location specific, protocol dependent, and by being tightly coupled. Jini avoids these pitfalls by providing a self-managing, self-configurable distributed network architecture that is based on providing services.

The federation of resources, devices and users in a Jini network is known as a djinn. Jini's normal discovery and join protocol uses a multicast request protocol interaction to find a nearby lookup service, followed by a unicast request protocol to obtain a client stub from the lookup service for a desired service. The multicast request protocol does not require an underlying Java Virtual Machine (JVM), as it is based on underlying network protocols [8]. Our work requires that an entity uses the multicast protocol to find and interact with an authentication server before becoming part of a djinn. After successful authentication, contact with a lookup service can be initiated.

### 2.2 Certification Authorities

A certificate authority is tasked with verifying the identity of entities, issuing public and private key pairs, and digitally signing the public key to assure that the identity is tied to the key pair. We have implemented certification authorities that mimic the hierarchical structure of an example organization

employing the ad hoc network. Our example organization has a parent corporation, which has divisions within the corporation. Within each division are departments that support the division's goals. Groups are the lowest working unit within the departments. Ad hoc networking would be an asset for data sharing during strategic meetings with participants from outside the local group. An organization requires the flexibility to have someone from another department, as well as someone from the corporate office, participate in a network and share resources. However, that flexibility should not allow unprotected access to company resources and data. An unauthorized computer within the range of the network should not be able to join the network by exploiting the dividends of eavesdropping. The authorized entities participating are assumed to be members of the same certification authority tree with certification authorities (CAs) available at each organizational element. Figure 1 depicts the certification tree for our example, with the parent company being the root CA00, the divisions having the next level of authorities CA10 and CA11, and so on.

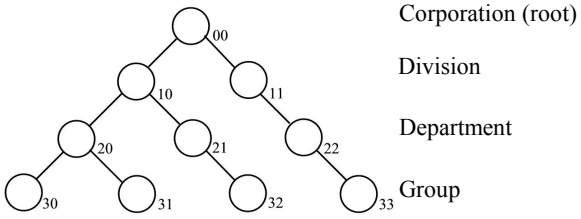


Figure 1. Certification Tree.

Each entity has membership in each of its domain levels, such as department, division or company. Two entities from different groups will have to use the shared CA closest to their level for authentication. In this case, the closest CA the entities share is the department CA. The following discussion addresses entity (e.g. computer) authentication; it is assumed that users of the network entity have been authenticated via means such as biometrics or dynamic authentication [10]. The CA workstation used to generate RSA key pairs must be physically secure, and key exchange must be through secure means. Once a user has received a key pair, the CA must delete information about the private key [2].

In a conventional PKI implementation, requests for a public key are made to the X.500 directory, or by querying a Policy Certification Authority (PCA)[2]. In the dynamic environment of an ad hoc network, there is no central authority to provide key lookup. The method to establish the validity of a key must reside in each entity on the network.

### 2.3 Mobile Adhoc Network Security

Mobile, ad hoc, networks (MANETs) have dynamic topology, and since entities are continuously joining and exiting the network, the degree of effort necessary to secure the network is dramatically increased. Various attempts at addressing security in MANETs have been considered, including the absence of security, simple "shared-secret" authentication, and full PKI implementation. Current standards, such as IEEE 802.11, implement the symmetric key approach to securing communication. Use of a symmetric key reduces management complexity and encourages faster connection establishment. However, symmetric key solutions do not provide strong security and are easily susceptible to eavesdropping. Conversely, PKI implementations in MANETs could be extremely cumbersome. Public key implementations are considerably more resource intensive than symmetric ones, a critical consideration given the limited capabilities of MANET nodes. There is no obvious solution for supporting key management services like certification, revocation, and key issue due to the numerous possibilities for insecurities [3, 11, 12].

Bluetooth wireless technology supplies an ad hoc connection that allows many heterogeneous units to communicate in a localized ad hoc network. The proprietary Bluetooth algorithm for joining the network is based on a challenge-response authentication scheme. Bluetooth devices use a symmetric key that is arranged *apriori*; authentication is based on the challenger's ability to produce this key. A weakness lies in the assumption that the symmetric authentication key will remain secret among valid Bluetooth devices. While Bluetooth may be secure enough for small applications in the home, its ability to provide confidence on any large, sensitive network is questionable [6].

## 3 INCORPORATING PKI INTO A JINI-BASED NETWORK

Since any restriction will inhibit rapid connectivity in ad hoc networks, an authentication technique should be distributed and not require trusted third-party references at the time a request to join occurs. Therefore, we propose use of a distributed public-key authentication scheme, similar to Kerberos [13].

Our authentication algorithm assumes that every participant on the network participates within a PKI and in a hierarchal Certification Authority (CA) structure. We also assume each public key in the system has been signed by the immediately higher authority's private key. In turn, each signing CA's public key has been signed by a higher authority's private key, with the last signer in the chain being the root CA. Additionally, we assume that the chain of certificates attached to a public key has been issued to each entity before that entity attempts to connect to an ad-hoc network. A successful authentication results from a proof of identity, accomplished by ascertaining that a common certification authority exists in the joiner's and authenticator's chain of certificates.

### 3.1 The Certification Authority Structure

In our certification process, the original certificate authority is the parent company. This authority signs the certificate of an individual division's public key using the root corporation's private key. The division similarly signs keys of its department CAs. The chain continues until each group CA has signed the keys of the individual entities in the group. It is assumed that each entity has the keys for its own authorities in its chain, allowing rapid comparisons with keys they receive from other platforms in the same certification chain. While key generation and certificate signing is hierarchical, the use of the certificate chain allows any entity to authenticate another party's key by searching for a common authorized signature. The determination of the validity of a key not in the certification chain requires starting at the common closest level, from the group CA, up the certificate chain to find a link that can be decrypted with a key from its own CA at the same level. Once a common CA has been found, the entity attempting to authenticate another can trust the common CA signature on the next (unknown) CA below

it. The process continues until the CA signing the joining entity's key genuineness has been established, thus founding the authenticity of the joiner's signed key.

Figure 2 depicts a scenario in which an entity, A, is acting independently, and discovers an ad hoc network via a multicast-discovery protocol. Entity A begins authentication by contacting entity B, which is already a member of the network. To guarantee authentication, each member participates in a distributed, public-key protocol. In this case, the closest CA that entities A and B share in their certificate chain is CA2. Since CA2 is trusted, and has signed the CA below it, it follows that CA5 can also be trusted. At that point, the joining entity's key can be used.

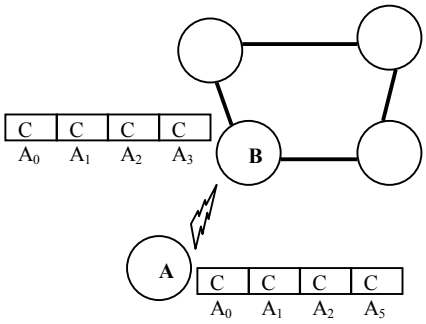


Figure 2. Ad Hoc Certificate Validation.

Our implementation requires a cryptographic library to provide asymmetric encryption support. Since Jini does not support cryptographic algorithms, we investigated several options for implementing the encryption and decryption functions needed, including the "native" mechanism of Jini [8] to call cryptographic libraries available in C++. While these libraries provided the asymmetric encryption and decryption to support PKI, none was easily adaptable to constructing a certificate chain of signed keys. Sun's Java Cryptographic Extensions (JCE) library [7] allowed the use of asymmetric algorithms such as RSA to encrypt and decrypt arbitrary data, and Legion of the Bouncy Castle release 1.11 was used to generate RSA keys [9]. We then applied the JCE cryptographic functions were used to build the PKI elements of the authentication implementation.

### 3.2 A Java-based Certification Authority

Our approach requires that each key be signed only by its immediate certification authority. The certificate chain that B holds contains a series of public keys. Each key is accompanied by a data structure containing the administrative information about the certification authority, all encrypted by a superior CA. When one of B's public keys is able to decrypt a link in the chain, the result will be the public key needed to decrypt the next link. Using this process, once B successfully decrypts A's chain of public keys, B is able to obtain A's public key.

A data structure containing administrative data is appended to each key. In a fielded system, the first entry in the data structure is a character string that will be regularly changed by the CA. Since the data structure is appended in plain text, and is encrypted with a private key, there exists the potential for cryptanalysis. However, if the first entry of the data structure is dynamic, there will be less obvious differences between the end of the public key and the plaintext data, making cryptanalysis more difficult. Figure 3 shows the certificate chain, and the details of a link in that chain.

## 4 JOINING PHASE EXPERIMENTS

In this section, we examine the specific steps of the joining phase of our algorithm, and analyze the data we collected during use of our joining algorithm. We consider the actual run times of our implementation in terms of key extraction, and evaluate a Java application that provides a benchmark for comparison

### 4.1 The Joining Phase

We implemented our joining algorithm using Sun Microsystems' Jini Technology version 1.1 [1]. Our joining algorithm is divided into three distinct phases: initialization, discovery, and authentication. As shown in Figure 4, the initialization phase requires that the authentication service

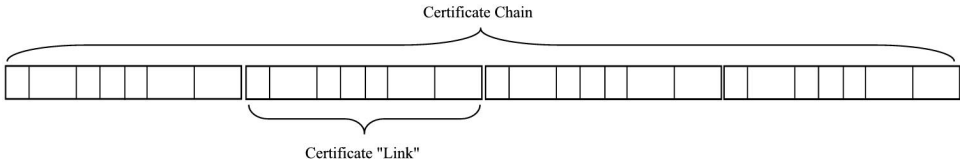


Figure 3a. Certification Chain (adapted from [5]).

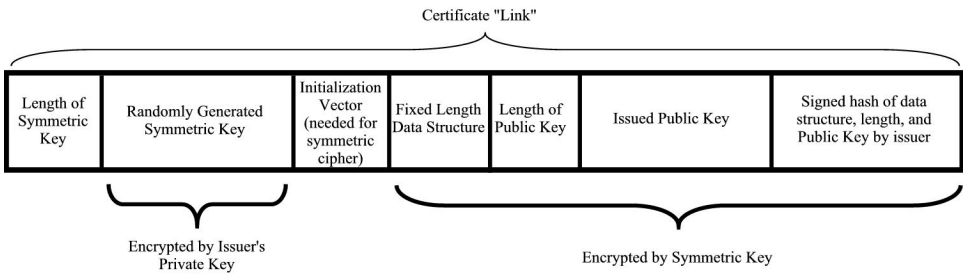


Figure 3b. Link in the Certification Chain.

provider register itself with a lookup service. A joining entity will search for, and locate, a network during the discovery phase. To focus our discussion of operations during this phase, we assume that the network exists, and that the joining entity successfully discovers the network. Although the actual discovery protocol is independent of our joining algorithm, we currently use Jini's multicast protocol for discovery.

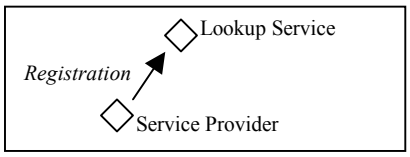


Figure 4. Initialization Phase of the Joining Algorithm.

As discussed in our previous work [4], there is a four part procedure that the joining entity must perform before it can fully participate in an ad hoc network. This procedure occurs after the multicast portion of discovery has been completed, and the lookup server has been found. The procedure is described below, and shown in Fig. 5.

1. Obtain Proxy. The lookup service provider provides a proxy to the joining entity (client) to use for communicating with the authentication service provider.
2. Client Sends Certificate. After the client has obtained the service's proxy, its public key and certificate chain is sent to the authentication service provider. The service provider computes the validity of the certificate and decides whether or not to authenticate the client.
3. Authentication Service Acknowledgement. The authentication service provider sends an acknowledgement containing the shared symmetric key to the joining entity.
4. Acceptance. After the joining entity has been positively authenticated, it may participate on the network. Possession of the shared symmetric key allows the entity to communicate and interact with the other entities on the network. The authenticated entity may register as a service, or use network services.

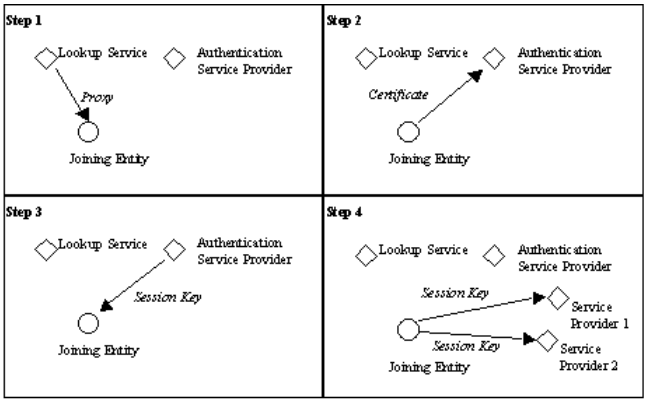


Figure 5. The Steps to Authentication.

### 4.2 Experiments

We implemented our algorithm using Jini 1.1 technology. Jini offers automated synchronization of network resources that are critical to dynamic networking. A Jini service is ubiquitous in the network, making it readily available to the client. Using Jini technology, the network provides the user with the desired service by first locating and then installing the necessary components. We modeled the authentication procedure as a Jini service. The authentication service determines if a client may be allowed to access the network, therefore the client may interact with it before contacting any other network service providers. Before a client can access the authentication service, the service itself must register itself with a lookup service. The service object uploads its proxy to the lookup service; the lookup service supplies that proxy to clients. When an object requests authentication from the network, the lookup service responds by sending the requester the downloadable proxy needed to communicate with the authentication service. Using that proxy, the joining object then contacts the authentication server directly. This portion of the joining algorithm is implemented in Jini by having the joining object use the authentication server's published proxy to initiate contact.

Our efforts began by placing the joining entity and the authentication service provider objects on one physical workstation to verify our implementation without the influence of network. The next step was to move the objects to different workstations connected by an 802.11 network.

#### 4.2.1 Run Time of the Algorithm

There was considerable set up time initiating the software for the experiments, so each of the four scenario cases was run six times. These set up times were an artifact of our experiments and would not be a factor in a running system. The resulting times are shown in Table 1.

Table 1. Total Algorithm Running Times (ms).

| Same Group | Same Division | Same Department | Same Corporation |
|---|---|---|---|
| 7407 | 8454 | 7052 | 7896 |
| 8516 | 7477 | 8933 | 17131 |
| 8411 | 9935 | 8389 | 9751 |
| 11607 | 6729 | 7881 | 9341 |
| 7866 | 6718 | 9173 | 10925 |
| 10724 | 7045 | 7078 | 10564 |
| 9089 | 8136 | 8084 | 10935 |

#### 4.2.2 Run Time Key Extraction

Since the algorithm running times were not linear, as we had originally predicted, we measured the time required for certificate extraction. The certificate extraction times do display linear growth as an increasing number of keys are extracted as shown in Table 2, and graphed in Figure 6.

Table 2. Certificate Extraction Time vs. Total Run Time (ms).

| Same Group | Same Division | Same Department | Same Corporation |
|---|---|---|---|
| 362 | 439 | 470 | 586 |
| 344 | 427 | 508 | 590 |
| 359 | 453 | 491 | 578 |
| 371 | 415 | 486 | 588 |
| 359 | 434 | 489 | 586 |

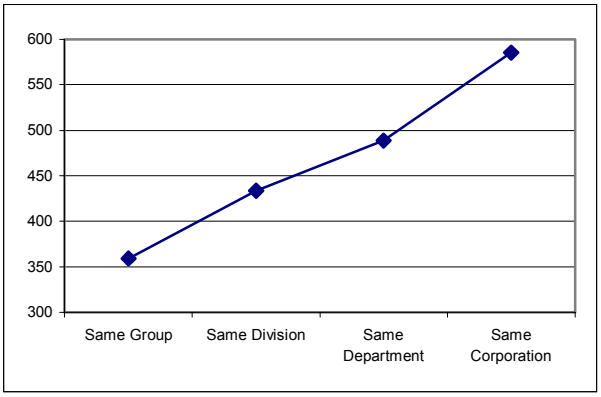


Figure 6. Certificate Extraction Times (ms).

Our next effort was to investigate the causes of the non-linear times measured when the whole authentication algorithm was running. The two best candidates for causing non-linearity were the Java Runtime Environment (JRE) itself and the network communications that our algorithm required.

#### 4.2.3 Benchmark with Java Runtime Environment

When a Java application runs continuously, it consumes system resources, potentially causing a Java application's performance to suffer. To test this effect, we ran a benchmark program that performs that mathematical operation, $temp = \sqrt{\sqrt{10} + \sqrt{10}}$. Two hundred loops of two million iterations of the operation were performed. The results are shown in Figure 7.

#### 4.2.4 Benchmark with RMID

Next, we ran the same Java benchmark while the RMI daemon (RMID), needed for Jini's network communications, was running. The resulting graph, in Figure 8, shows a peak at 1.1 second, almost five times longer than the benchmark without RMID running. The performance under these conditions is more irregular, indicating that RMID is affecting the performance of our whole algorithm.

We concluded that RMID has a greater impact on the run time than the normal JRE. Although RMID is running while the certificate extraction occurs, we concluded that the server performs the certificate extraction without using the RMI protocol for network communication.

## 5 ANALYSIS

The Jini implementation supported our algorithm's design goal of allowing distributed, ad hoc, and net-centric connections. The location and procedures of the authentication service was transparent to client. The client was able to access the service through a simplified interface without *apriori* knowledge of the service's location, and without knowledge that the service even existed. While there is overhead incurred by requiring authentication prior to allowing a client to contact a service, the delay will be offset by the increased level of security that has been introduced.

Although the RMI daemon influences the resulting authentication times, our experiments show that the certification extraction process remains roughly linear. Certificate extraction depends only on where the common Certificate Authority signature is found in the chain.

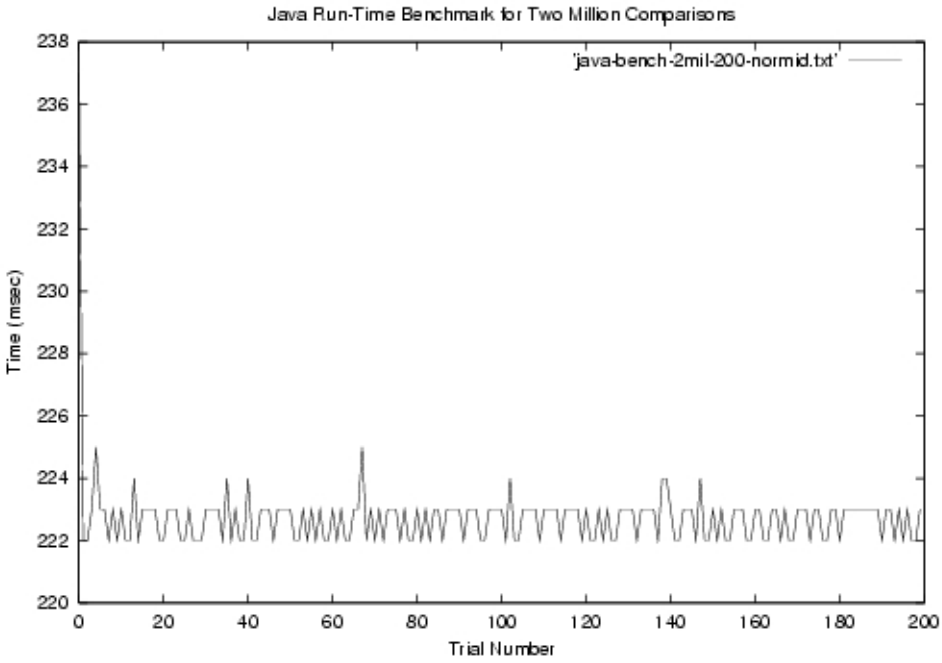


Figure 7. JRE Benchmark.

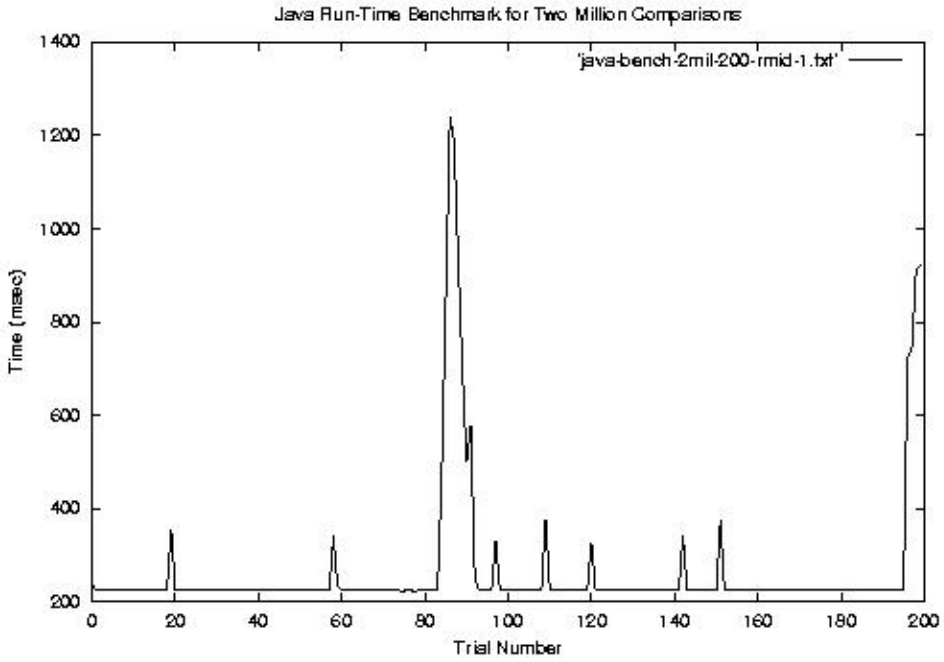


Figure 8. JRE Benchmark with RMID Running.

## 6 CONCLUSIONS

We discussed the need for a distributed public key protocol, and presented an algorithm that facilitates wireless ad hoc joining requests with the key assumption that an *apriori* public-key infrastructure is in place. We expect that the authentication time will grow larger as distances or number of users increase; therefore the success of the algorithm is affected by the same factors that affect wireless communications in general.

## 7 FUTURE WORK

We are currently rehosting the authentication software to the Windows Operating System. We plan to extend the effort to potentially minimize the impact of RMID. We also plan to study the performance of the network services after the authentication process is complete.

## ACKNOWLEDGEMENTS

This work was supported in part by the United States Naval Academy under the Trident Scholar Program.



## REFERENCES

[1] K. Arnold, The Jini Specifications, 2nd ed, Addison-Wesley, 2000.

[2] N. Berge, "UNINETT PCA Policy Statements", RFC 1875, Network Working Group, Dec 1995.

[3] S. Corson, and J. Macker, “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations”, RFC 2501, Network Working Group, Jan 1999.

[4] M. M. McMahon, J.D. Datko, and D. M. Needham, "An Authentication Scheme for a Jini-based Ad Hoc Network", 15th International Conference on Parallel and Distributed Computing Systems, Louisville, KY, pp 462-477, September 2002.

[5] J. Garms, and D. Somerfield, Professional Java Security, Wrox.Press, Birmingham, UK, 2001.

[6] G. Held, Data over Wireless Networks : Bluetooth, Wap, and Wireless LANs, McGraw-Hill, 2000.

[7] Java Cryptographic Extensions (JCE) library available from http://java.sun.com/products/jce/

[8] S. Kumaran, Jini Technology: An Overview, Prentice Hall PTR, 2002.

[9] Legion of the Bouncy Castle, Release 1.11, available from http://www.bouncycastle.org/latest_reease.html

[10] M. M. McMahon, H. A.Sholl, and R.A.Ammar, “Proposed Structure of a Network Security System Using Dynamic Authentication”, Proceedings of the Parallel and Distributed Computing Conference, Dallas, TX, pp 183-189 Aug 2001.

[11] C. Perkins, Ad Hoc Networking, Addison-Wesley, New York, 2001.

[12] B. Schneier, “802.11 Security” Crypto-Gram Newsletter, March 15, 2001.

[13] M. Sirbu, “Distributed Authentication in Kerberos Using Public Key Cryptography”, Information Networking Institute’s Net Bill, Carnegie Mellon University, 2001.