Lightweight security for JXME-proxied relay authentication

***Abstract*—Mobile devices have become ubiquitous, allowing the integration of new information from a big range of objects. But the development of new applications requires a powerful framework which simplifies their construction. JXME is the JXTA protocols implementation for mobile devices using J2ME, its main value being its simplicity when creating peer-to-peer (P2P) applications on limited devices. On that regard,an issue that is becoming very important in the recent times is being able to provide a security baseline to such applications. This paper proposes a simple security mechanisms in order to protect JXME applications against a broad range of vulnerabilities. The protocol overhead has been experimentally tested in order to assess its low impact on device performance, an important requisite on limited devices.**

***Keywords*-peer-to-peer; security; JXME; JXTA; distributed systems; Java; J2ME.**

I. Introduction

Peer-to-peer (P2P) applications have become highly popular in recent times due to its great potential to scale and the lack of a central point of failure. Slowly, they have evolved from simple file-sharing environments, such as Gnutella [1], to more complex ones such as e-health or e-learning [2]. However, currently, Internet is quickly becoming witness to the transition from a desktop-centric environment towards one based on the ubiquity of mobile devices [3]. Therefore, it was natural that the next step in the evolution of P2P applications would be following this trend [4], since mobile environments are based on node autonomy and decentralization, just like P2P.

There are different platforms that allow programmers to develop mobile P2P applications [5], [6], among which JXME [7] can be found. Its main advantage, in comparison with other proposals, is being the mobile version of the well known JXTA platform [8]. The JXTA specification defines a set of generic protocols which allow peers to communicate and publish, find or consume remote resources, independently of the actual transport layer and the implementation language. Such protocols are generic enough so they are not bound to a narrow application scope, but adaptable to a large set of application types. Nevertheless, JXTA was designed with desktop devices in mind. Thus, JXME was developed in order to allow mobile devices to create standalone mobile JXTA networks or to participate in a standard JXTA network using a mobile device.

The main characteristic of JXME is that it seriously takes into account the fact that a transition from a desktop environment to a mobile one requires facing challenges such as maintaining the trade off between scalability and efficiency, as well as the idiosyncrasies of mobile devices, such as power and storage limitations. However, the maturity of research in the field of P2P and mobile environments has pushed through new problems often neglected in a framework's design: those related with security. Even under the constraints of limited devices, a security baseline must be kept in any P2P system in order to protect it against common network vulnerabilities.

The purpose in this paper is proposing an improvement to the security baseline in the simplest version of JXME, named JXME-Proxied. The proposed security mechanism guarantees lightweight authentication of peers towards their relay peer, taking into account the limitations in terms of resources in mobile devices running JXME. The protocol has been executed under an experimental testbed in order to evaluate its overhead, since this is a important constraint in limited devices.

This paper is structured as follows. Section II provides an overview of JXME and summarizes its current related research. In Section III a simple security mechanism for JXME-Proxied is proposed. Section IV shows the experimental results obtained from evaluating the impact of the previously presented security mechanism. Finally, Section V summarizes the paper's main contributions and outlines further work.

II. JXME overview

JXME is based on JXTA, sharing the same main specification. The basic organizational foundation in both JXTA and JXME is the *Peer Group*, a set of peers with common interests which agree on common services. By default all peers belong to the *NetPeerGroup*, a boot Peer group which has a well know identifier, since it is hardcoded in the JXME source code.

The *Membership Service*, one of JXTA's core services, manages the group members' identities within a Peer Group. In addition, this service is defined as generic in the JXTA specification, allowing developers to implement their own version with the security schemes required by their applications. Before joining a Peer Group, identities are assigned by successfully completing an authentication process.

Any resource may be shared between Peer Group members by distributing its associated *Advertisement*, an XML metadata document describing the resource properties and how it may be accessed. The *Discovery Service* manages Advertisement location and distribution. Every time an Advertisement is retrieved by a peer, it is stored in a local

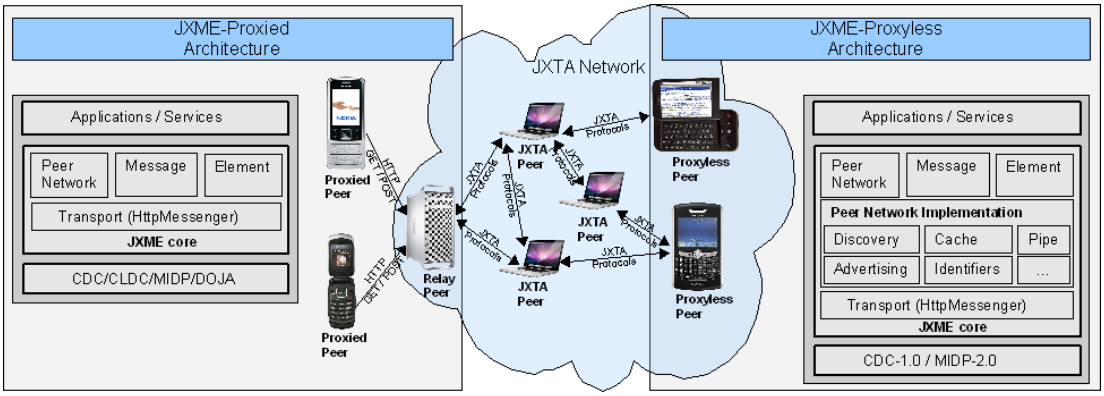

Figure 1. JXME architecture

cache and assigned an expiration date. Advertisements will be automatically flushed when its expiration date is reached. After locating a resource, such as a service, messaging may begin using JXTA *pipes*, abstract endpoints which provide an asynchronous unidirectional communication channel.

As far as JXME is concerned, it is based on two framework specifications for Java ME: *Connected Device Configuration* (CDC) and *Connected Limited Device Configuration* (CLDC). The former uses the C-Virtual Machine (CVM), an optimized version of the Java Virtual Machine (JVM), has most of the standard Java classes and is addressed to powerful devices. On the contrary, the latter uses the Kilobyte Virtual Machine (KVM) [9], has few of the standard Java packages and is addressed to devices with limited resources in terms of processing and memory capacity. Moreover, CLDC has two operation modes defined: *Mobile Information Device Profile* (MIDP) and *DOcomo JAva* (DOJA). MIDP is a specification for the usage of Java on embedded devices and DOJA is a Java environment specification for DoCoMo's i-mode mobile phone.

JXME has two distinct versions suitable for different device types, according to their capability limitations. On one hand, the *JXME-Proxied* version is a simple implementation for very limited devices, which delegates all heavyweight work to an external super-peer. On the other hand, the *JXME-Proxyless* version is a more complex one, where peers may directly interact with the JXTA network. Figure 1 shows the JXME-Proxied and JXME-Proxyless main components and how they interact with a JXTA network.

In this paper we will focus on a security mechanism for the JXME-Proxied version since it is the most thoroughly tested and used [10]. Furthermore, its basic architectural design departs from JXTA in some basic aspects, which make it difficult to just adapt and deploy security mechanisms which already exist in JXTA but have not been implemented. New approaches must be considered.

A. JXME-Proxied

The JXME-Proxied version is the simplest and the oldest one. This version is implemented in Java ME for both CDC and CLDC framework specifications, as well as for the MIDP and DOJA profiles. Devices which use JXME-Proxied in a JXTA network are named *Proxied Peers*, and cannot directly interact with other peers within the JXTA network. All communications from a Proxied Peer are actually destined to a super-peer which will overcome the limited capabilities of a Proxied Peer by translating or summarizing requests and responding to queries in its behalf. This special type of super-peer is called *Relay Peer*, implementing the Relay and Proxy JXTA services. The former is used in JXTA networks where there is no direct connectivity between peers, to traverse firewalls or NATs. The latter allows access to peers which do not implement all of the JXTA basic services. The communication between the Proxied and Relay Peer is performed with a simplified protocol based on HTTP. By default, a single Relay Peer can support up to 150 Proxied Peers.

The main responsibilities of the Relay Peer on regards to its Proxied Peers are:

- Listen to and answer requests from the Proxied Peer.
- Translate to XML messages received from the Proxied Peer and retransmit them to the JXTA network.
- Store messages received from the JXTA network for the Proxied Peer. Such messages are only actually retransmitted when polled by the Proxied Peer.
- Summarize and translate XML messages from the JXTA network into a simple format which the Proxied Peer is able to understand.

The main components of JXME-Proxied can be identified in Figure 1. Due to its limitations and reliance on a Relay Peer, the kind of operations that a Proxied Peer can actually execute are limited: **Join** a group, **Search** or **Create** resources (such as Peer Groups or pipes), **Listen** to a pipe to receive data, **Send** data to a specific pipe, **Close** a pipe and **Poll** the Relay Peer for messages from the JXTA network

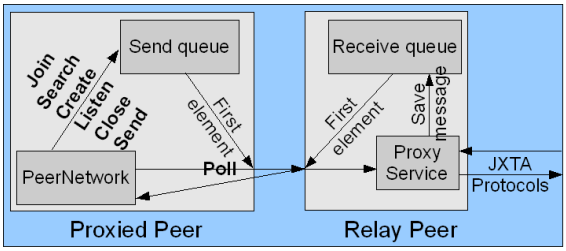

Figure 2. Proxied communication scheme

that have the Proxied Peer as the final destination.

The communication between the Proxied and Relay Peer follows a simple protocol using HTTP-POST encapsulation. A Message, formed by a set of Elements, is created whenever an operation is executed and sent to the Relay Peer. Every Proxied Peer queues ongoing messages, which are actually sent at a set interval, using the poll operation. Thus, communication time is restricted to specific periods.

Figure 2 shows the communication scheme between a Proxied Peer and its Relay Peer. In this scheme, two communication types can be identified, one between the Proxied Peer and the Relay Peer and another between the Relay Peer and the JXTA network. In the former a Proxied Peer uses a local identifier within the Relay Peer context, a local PeerId (from now on, we will refer to it as just PeerId). In the latter this identifier is translated to a global identifier for the JXTA network, a JXTA PeerId.

The standard JXME-Proxied general operation cycle includes the same steps as in JXTA:

1) **Platform startup**: This is the first action performed by a JXME Peer and consists in loading the required libraries. Then, the peer associates with a Relay Peer, and a internal and JXTA PeerId are created.
2) **Peer Group joining**: The peer joins a Peer Group and from then on, is able to interact with other Peer Group members. This stage is managed by the Membership Service, just like in standard JXTA.
3) **Resource discovery and publication**: Encompasses the distribution of Advertisements. A resource cannot be accessed without previously retrieving its Advertisement.
4) **Message exchange**: This is the most frequently action performed by Proxied Peers. A Proxied Peer has two operations available in order to exchange messages: listen and send.
5) **Disconnection**: This is the last step a peer performs in order to exit the JXTA network and disconnect.

After this overview of the JXME-Proxied version, it is obvious that the need of Relay Peers is the main design divergence and limitation of this approach from original JXTA. The main consequences are twofold. First of all, if a set of Proxied Peers join the network using a single Relay Peer, then a central point of failure is created for all of them. However, and secondly, if a single Proxied Peer simultaneously connects via different Relay Peers in order to avoid the former pitfall, it will be assigned a different JXTA PeerId by each Relay Peer. Thus, the Proxied Peer, by simultaneously having different identities, will be considered as several different peers within the JXTA network.

B. Related work

Different proposals in the literature on JXME are reviewed in this section. The objectives of conducting this review are, first of all, identifying noteworthy studies or proposals about JXME, and secondly, looking for security proposals.

A comprehensive study on P2P applications, developed with JXME, is detailed in [10]. Piedrahita and Montoya present a testing model to be able to measure some aspects of the JXME system. The authors consider that JXME for CLDC (JXME-Proxied) cannot be completely considered a P2P system, since a full dependency on the Relay Peer exists. If the Relay Peer fails, the system does not have the ability to automatically choose another one for the application to continue working.

Other applications for JXME are found in [11] and [12]. In the former, Blundo et al. propose JXBT as an implementation of the JXTA/JXME infrastructure using Bluetooth as the communication channel. In the latter, Tahsin et al. present two P2P demo applications for mobile devices based on JXME: a Message Passing application and a File Sharing application. The goal of these two applications was to show that it is possible to develop a P2P application for mobile devices capable of identifying and transferring large files without depending on a central server.

A framework for mobile devices optimized to MANET networks which is compatible with JXTA protocols is developed in [13], where devices interact with each other in an ad-hoc style. Furthermore, authors analyze the performance overhead introduced by maintaining interoperability with JXTA protocols. Their concussion is that the small overhead introduced by using XML is acceptable because of the advantages that be interoperable with JXTA protocols provides.

As far as security is concerned, Kawulok et al. [14] show a framework which allows wireless and remote peers to participate in a JXTA network. Authors describe the most interesting implementation details of the framework as well as all changes made in the JXTA core and JXME packages. The proposed framework adds a new authentication scheme based on certificates and PKI [15]. This authentication is provided by the Relay Peer, which uses an external Sign and LDAP Server, breaking completely the P2P model proposed by JXTA.

In [16] a security analysis of JXME-Proxied has been conducted. Authors highlight that the work done in JXME security is very poor and conclude that this version does

not have an appropriate security baseline. The main reason for this is the fact that, since messages are exchanged with the Relay Peer in clear text and no authentication is provided, many different attacks are very easy to pull off. For instance, any Proxied Peer is easily able to impersonate another Proxied Peer by just directly usurping its PeerId, or use its open pipes to send messages on its behalf.

From this literature review, it can be concluded that there are several publications and projects about JXME, but, unfortunately, there are few contributions to JXME security. Also, the security baseline provided by JXME-Proxied is found to be insufficient.

III. Improving JXME-Proxied security

The security analysis presented in [16] shows that the security mechanisms provided by JXME-Proxied are still not sufficient to secure standard mobile applications. This is because the current version is vulnerable to a wide range of attacks. However, some attacks can be prevented by simple schemes that extend the basic protocols used in JXME-Proxied, adding only a bounded complexity. In particular, we present a proposal to avoid *Spoofing* and *Replay* attacks. It is important to highlight that this new mechanism does not try to solve every single vulnerability which was identified, but provides a initial protection in the communication by guaranteeing lightweight authentication. This is the first step in providing a secure mobile framework.

A. JXME-Proxied vulnerabilty overview

The main vulnerabilities found in JXME-Proxied are produced by the insecure communication link between the Proxied Peer and its associated Relay Peer. For that reason, the proposed security scheme tries to secure the common message protocol over HTTP between the Proxied and Relay Peer. A secure extension for this communication protocol is built. Under the assumption that a Proxied Peer is a very limited device, the proposed protocol extensions are based on basic lightweight cryptographic operations, mainly hash functions.

Protection against spoofing and replay attacks may be obtained in a straightforward manner by securely identifying the Proxied Peer through any well known authentication scheme. However, this proposal pretends to be as simple as possible since the constrained resources of devices where Proxied Peers are executed must be taken into account. For that reason, the protection against spoofing and replay is obtained not by linking messages to a particular peer identity but by guaranteeing that, given a set of messages, all come from the same source peer, whichever that source might be. Thus, once an identifier is assigned, it can be guaranteed no intruder is able to insert false messages impersonating the source peer. To achieve such goal, we propose that a hash-chain [17] based scheme is used to create a set of linked values which will be used as local identifiers.

In this scheme, is really important to understand in detail how a Proxied Peer performs the platform startup stage, described in Section II-A. A Proxied Peer first loads the platform binaries and before it can join the JXTA network, it must associate to any available Relay Peer. According to the JXME specification, during this process, the Proxied Peer requests a new identifier, *PeerId*, which, according to the JXME specification, should be generated by the Relay Peer and sent back to the Proxied Peer. Such identifier is only used within the context of Proxied Peer and Relay Peer message exchange and has no prevalence in the general JXTA network. Thus, the PeerId generation process is very important, since the Relay Peer is only able to identify Proxied Peers by their PeerId. However, it was found that in the actual implementation Proxied Peers can freely generate their own PeerId and connect to the Relay Peer, skipping most of this process. In fact, due to this occurrence, any peer may trivially impersonate others by self-assigning a PeerId already in use.

The PeerId generation process is an exception to the JXME-Proxied message format since the request is sent using HTTP-GET. An example of an identifier request message is shown in Listing 1. It can be recognized as such since the PeerId is specified as `unknown-unknown` in the GET command.

```
GET /unknown-unknown?0,-1,http://192.168.0.37:2481/
EndpointService:jxta-NetGroup/uuid-DEAD...05/pid HTTP/1.1
Connection: close Content-Length: 0
User-Agent: UNTRUSTED/1.0
Host:172.16.0.37:2481
```

Listing 1: - Proxied Peer PeerId request message

Therefore, we propose a security extension to the basic PeerId generation protocol at the platform startup stage in order to counter attacks regarding authenticity at every stage in a peer's lifecycle. When the PeerId is obtained, Proxied Peers generate a sufficiently long hash-chain (taking into account available resources) and use each intermediate value as its PeerId in each successive message exchange with its associated Relay Peer. In this way, the PeerId attached in a message changes for each successive message in a manner that cannot be predicted by a possible attacker. However, the Relay Peer will be able to easily track identifier changes and recognize each message as originating from the same source. Using a changing PeerId allows us to use exactly the same original protocol format, without the need to add additional fields.

B. Protocol initialization

The proposed scheme needs and initialization process executed during the Proxied Peer startup step and boot operation. In this process, the hash-chain is created, the values are stored in the Proxied Peer internal memory and

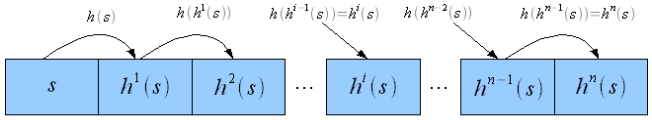

Figure 3. Hash-Chain creation

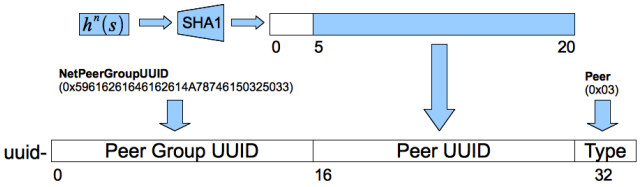

Figure 4. initId generation

the first PeerId is transmitted to the Relay Peer. Then, the communication messages between the Proxied Peer and the Relay Peer are executed using each successive PeerId from the generated hash-chain. The verification process allows the Relay Peer to ensure that all messages come from the same Proxied Peer. Such verification can be performed by computing one hash value and comparing it with the one obtained in the previous messages.

The detailed initialization process is next described:

1) At the startup stage, the Proxied Peer chooses a random seed, s.
2) The Proxied Peer generates a hash-chain $hc(s) = \{h^0(s), \cdots, h^n(s)\}$ by iteratively applying n times the hash function $h(\cdot)$ on s, so that $h^0(s) = s$ and $h^i(s) = h(h^{i-1}(s))$. All values in hc are stored in the peers' local memory. Figure 3 summarizes this process.
3) The Proxied Peer's initial PeerId, $initId$, is created from $h^n(s)$. A PeerId may be created from any value in $h^n(s)$ according to the following steps (summarized in Figure 4). The result fulfills the JXTA peer identifier specification.
   - The PeerId starts with the string $uuid-$.
   - The 16 most significant bytes, $\text{msb}_{16}(initId)$ or Peer Group UUID, are the *NetPeerGroup* identifier.
   - From the 16^{th} to the 31^{st} byte the Peer UUID (from now on, summarized as as PUUID) is specified. The 16 least significant bytes of the hash value, $lsb_{16}(h^n(s))$, are assigned.
   - The 32^{th} byte describes the ID type identifier. In this case, being a PeerId, the value 0x03 is assigned.
4) A PeerId request message is sent to the Relay Peer using its well-known address. However, instead containing an `unknown-unknown` string, as would be used in the original unsecure protocol, $initId$ is announced.

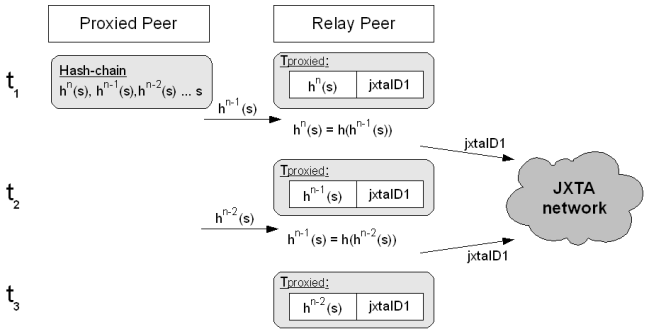

Figure 5. Hash-Chain consumption and identifier translation

5) When the request is received, the Relay Peer randomly generates a new JXTA PeerId, $jxtaID1$, as would be done in standard JXME. At this point, the Proxied Peer is considered associated to the Relay Peer.
6) The Relay Peer keeps track of the identifiers for its possible different Proxied Peers in a local translation table $T_{proxied}$ that contains two fields: `lastId` and `globalId`. The former contains PeerId's and the latter JXTA PeerId's. The Relay Peer translates local PeerId's to JXTA ones (as explained in Section II-A) when acting as some Proxied Peers behalf in the JXTA network. At this point a new entry is added to the table, `lastId` = $initId$ and `globalId` = $jxtaId$, where $initId$ is considered the entry's key.

C. Protocol execution

After the initialization process has been performed, secure communication between the Proxied Peer and the Relay Peer can begin. Each message will use a new PeerId generated from the successive values extracted from the $hc(s)$. These values are retrieved in the descending order from their generation, starting from $h^{n-1}(s)$ and ending in s. That is, in the second message, the Proxied Peer will use the identifier:

$$newId = "uuid-"||NetPeerGroupUUID||\text{lsb}_{16}(h^{n-1}(s))||"03"$$

In general, the j-th message between the Proxied and the Relay Peers will contain the identifier

$$newId = "uuid-"||NetPeerGroupUUID||\text{lsb}_{16}(h^{n-j+1}(s))||"03"$$

The identifier consumption from $hc(s)$ and the translation between the changing PeerId and the static JXTA PeerId is represented in Figure 5. To simplify this figure the identifier is represented as the PUUID part of the Proxied Peer identifier, which is the only one which varies at each message exchange.

The verification process is executed at the Relay Peer, validating that all successive messages come from the same Proxied Peer. This validation may be actually performed since, assuming that $lastPUUID$ is the PUUID section in

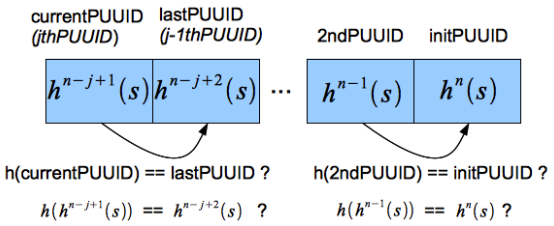

Figure 6. PUUID verification process performed by the Relay Peer

the identifier from the last message sent from the Proxied Peer (stored in the `lastId` field of $T_{proxied}$), then the PUUID section from the current message's identifier, $currentPUUID$, must hold true that

$$h(currentPUUID) = lastPUUID$$

.

The detailed verification process, for the PUUID of the j-th message can be seen in Figure 6 and its detailed explanation follows:

1) The Relay Peer takes the message identifier $currentId$.
2) The PUUID section of the identifier, $currentPUUID$, is extracted from $currentId$.
3) The Relay Peer calculates $h(currentPUUID)$. From this value a PeerId is generated following the steps described in Figure 4.
4) The result is looked up among the currently stored values in $T_{proxied}$'s `lastId` field.
5) If a match exists, the message is not a result of spoofing or a replay attack, since no other peer would be able to predict the currentPUUID ($h^{n-j+1}(s)$) from lastPUUID ($h^{n-j+2}(s)$) and use it as a portion of the message identifier. Only the legitimate hash-chain generator is able to calculate it, from its hash-chain.
6) The Relay Peer stores $currentId$ into $T_{proxied}$ replacing the old value matched in step 5. It becomes the entry's new key.
7) If, as a result of the received request, the Relay Peer needs to send messages towards the JXTA network on behalf of the Proxied Peer, the value stored in the $globalId$ field is used.

D. Hash-chain refresh

When a hash-chain is about to reach s, a new one must be generated and its initial value refreshed at the Relay Peer so the new hash-chain values may be used. s will be used as the PUUID part of the PeerId in the refresh message. To allow this process, the set of operations that a Proxied Peer can perform using HTTP-GET is extended with a `renew` command. This new parameter is used to announce a refresh in the hash-chain, containing the newId

$$"uuid-"||netPeerGroupUUID||h^n(newSeed)||"03"$$

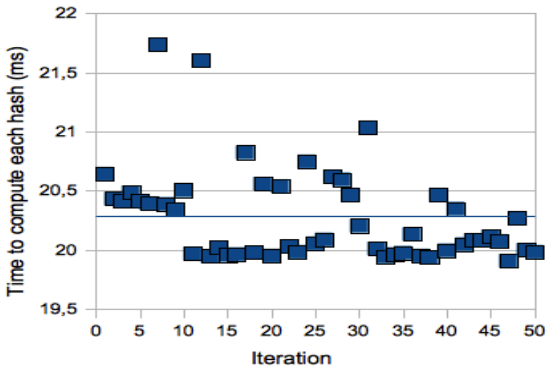

Figure 7. Mean time consumed by execute a hash operation

as its associated parameter.

When the Relay Peer receives any message which contains a `renew` command, in step 6 of the verification process, its content value ($newId$) is the one stored instead of the message's $currentId$, initializing the new set of local identifiers.

IV. Security overhead analysis

The actual implementation performance of our proposal has been evaluated by assessing how the protocol extensions added by this new security scheme would affect a Proxied Peer in terms of resource utilization. A mobile device acts as a Proxied Peer and a computer acts as a Relay Peer in these tests.

This mobile device needs network connectivity to exchange data with the Relay Peer and an open operating system which allows the execution of java applications and obtains information about the state of its resources, such as memory or battery usage. Based on these requirements, the device chosen was the HTC Hero. This is a popular mobile phone which is able to run JXME and has the following specifications:

- Processor: Qualcomm MSM7200A, 528 MHz
- Operating System: Android [3]
- Memory: ROM: 512 MB
- Memory: RAM: 288 MB
- Display: 3.2-inch TFT-LCD touch-sensitive screen with 320x480 HVGA resolution
- Network: HSPA/WCDMA: Up to 2 Mbps up-link and 7.2 Mbps down-link speeds
- Network: Wi-Fi: IEEE 802.11 b/g
- Battery: Rechargeable Lithium-ion battery with 3.7 V and 1350 mAh of capacity

The computer have to be powerful enough to be a Relay Peer, which is meet by most of today's computers. The chosen computer has been a laptop, a MacBook Pro.

The first test consisted in computing many hash operations, and then obtaining the average time required to

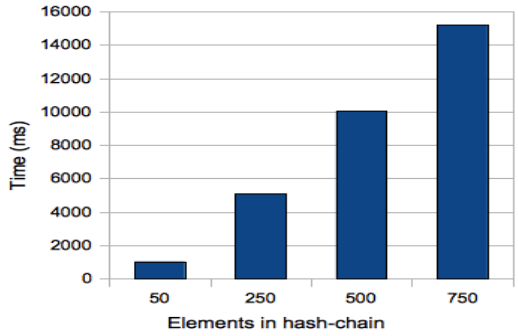

Figure 8. Time consumed by compute a hash-chain depending its size

HC size	min poll interval (s)	memory required (KB)
50	1	1,6
250	5	8
500	10	16
750	15	24

Table I
HASH-CHAIN OVERHEAD

compute each one. Figure 7 shows the average time that it takes to compute a single hash operation when computing hash-chains of 250 elements, and repeating this experiment 50 times. After this test, it is obtained that the average time it takes to compute a hash operation in a hash-chain of 250 elements is 20.28 ms. The values obtained during the 50 repetitions mainly fluctuate between 20 and 20.5 ms.

The previous test was focused in how long it takes to compute a single hash operation, but it is also important to know how long it will take to compute a hash-chain. The main reason is that while the hash-chain is being computed no communications between the Proxied and Relay Peer can be performed. Therefore, the size of the hash-chain will limit the minimum time between polls from the Proxied Peer to the Relay Peer. Figure 8 shows how long it takes to compute hash-chains of different sizes. It also shows that the time required to compute a hash-chain increments linearly when the size of the hash-chain increments.

Since Proxied Peer devices are expected to have limited resources, it is important to identify how much memory is occupied by the hash-chain, which is shown in Table I. The amount of memory required goes from 1,6KB when using a hash-chain of 50 elements to 24KB when using a hash-chain of 750 elements. This is respectively a 0.0005% and 0.008% of the total memory of the device used. In order to give some more perspective, for much more limited devices such as Sun SPOT [18], that would amount to 0.16% or 2.34% memory utilization.

From the data of Table I, Figure 9 is generated, showing how often the hash-chain will have to be renewed depending on its size and the poll interval time in seconds used. For instance, if an application requires a poll interval of

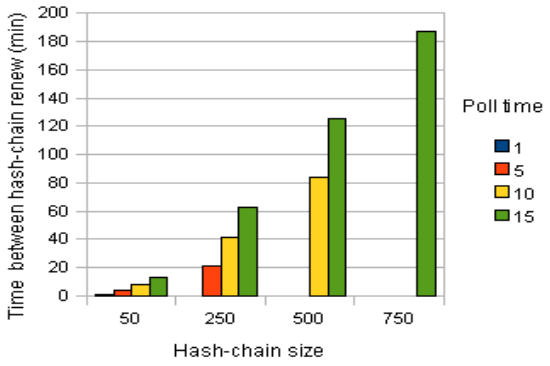

Figure 9. Time between hash-chain renews depending its size and poll interval time

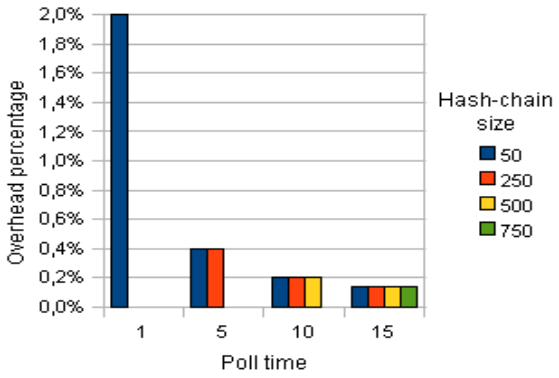

Figure 10. Overhead percentage depending of the size of the hash-chain and the poll interval time

5 seconds, the hash-chain can contain from 50 up to 250 elements. In the former, the memory required is 1.6KB and the hash-chain will expire each 4.17 minutes, whereas in the latter the memory required is a bit higher, 8KB, but it will last more time, 20.83 minutes.

Finally, Figure 10 shows the overhead percentage added by the periodically computation (renew) of the hash-chain depending on the hash-chain size and poll interval time. This overhead is calculated dividing the amount of time it takes to compute a hash-chain to the amount of time between renewing hash-chains. The results manifest that the overhead produced depends on the poll time interval and does not depend on the hash-chain size. For instance, using a poll time of 1 second the overhead will be of about 2% but using a poll time of 15 seconds the overhead will be just 0.13%.

V. CONCLUSIONS AND FURTHER WORK

The current version of JXME-Proxied is completely devoid of any security mechanism, thus being prone to all kinds of attacks. This paper proposes a simple security scheme for JXME-Proxied to solve the most glaring vulnerabilities which currently exist, providing basic protec-

tion against simple spoofing and replay attacks. A basic lightweight method based in hash-chains is chosen to extend the existing protocols, taking into account the idiosyncrasies of the devices which will host a JXME Peer.

The proposal minimizes the modifications of the JXME protocols, since no additional `request` type is defined. The `renew` command piggybacks inside any other naturally occurring request, such as a polling `listen` request, as an additional parameter, and will be processed along the original request. Therefore, there is no need to send a single message with the sole purpose of transmitting hash-chain data, reducing the overhead by taking advantage of existing transmissions. Furthermore, while no refresh is needed, the secure scheme does not even impact on the HTTP-GET protocol, since the peer identifier field is invisibly used, instead of using additional message element types.

In addition, we have run some experimental tests on JXME-Proxied so it is possible to evaluate the proposal and confirm that it is a light security mechanism in terms of resources consumption. Form the results, it can be concluded that the impact of applying the secure protocol is low, mainly due to its reliance on hash value computations.

Once an initial secure protocol extension has been established, further research includes providing additional lightweight security services to JXME which may counter other vulnerabilities. Furthermore, it is also important to compare these proposals to those provided by other middlewares for constrained devices, from both a qualitative and quantitative standpoint.

REFERENCES

[1] J. Frankel and T. Pepper, "Gnutella," `http://rfc-gnutella.sourceforge.net`, 2000.

[2] K. Matsuo, L. Barolli, F. Xhafa, A. Koyama, and A. Durresi, "Implementation of a JXTA-based P2P e-learning system and its performance evaluation," *International Journal of Web Information Systems*, vol. 4, no. 3, pp. 352–371, 2008.

[3] Google Inc., "Project Android'," `http://code.google.com/intl/es/android`, 2007.

[4] Skype, "Skype on your mobile," 2004, `http://www.skype.com/mobile`.

[5] G. Kortuem, "Proem: a middleware platform for mobile peer-to-peer computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 4, pp. 62–64, 2002.

[6] B. Christensen, "Experiences developing mobile P2P applications with lightpeers," *Peer-to-Peer Computing, IEEE International Conference on*, vol. 0, pp. 229–230, 2006.

[7] Sun Microsystems, "Project JXME," 2003, `https://jxta-jxme.dev.java.net`.

[8] Oracle, "Project JXTA," 2001, `http://www.jxta.org`.

[9] Sun Microsystems, "J2ME building blocks for mobile devices. white paper on KVM and the connected, limited device configuration (CLDC)," 2000, `http://java.sun.com/products/cldc/wp/`.

[10] T. Piedrahita and E. Montoya, "Performance analysis of JXTA/JXME applications in hybrid fixed/mobile environments," *Revista Colombiana De Computación*, vol. 7, no. 1, 2006.

[11] C. Blundo and E. D. Cristofaro, "A bluetooth-based JXME infrastructure," in *Lecture Notes in Computer Science*, vol. 4803/2009, 2009, pp. 667–682.

[12] T. Tahsin, L. Choudhury, and L. Rahman, "Peer-to-Peer mobile applications using JXTA/JXME," in *11th International Conference on Computer and Information Technology*, vol. 1, 2008, pp. 702–707.

[13] M. Bisignano, G. D. Modica, and O. Tomarchio, "JMobiPeer: a middeware for mobile peer-to-peer computing in MANETs," in *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pp. 785–791.

[14] L. Kawulok, K. Zielinski, and M. Jaeschke, "Trusted group membership service for JXME (JXTA4J2ME)," in *Wireless And Mobile Computing, Networking And Communications, (WiMob'2005), IEEE International Conference on*, vol. 4, Aug. 2005, pp. 116–121.

[15] "Internet x.509 public key infrastructure." 1999, `http://www.ietf.org/rfc/rfc2459.txt`.

[16] M. Domingo-Prieto, J. Arnedo-Moreno, and J. Herrera-Joancomarti, "Jxta security in mobile constrained devices," in *Advanced Information Networking and Applications Workshops, International Conference on*, vol. 0. IEEE Computer Society, 2010, pp. 139–144.

[17] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[18] Sun Microsystems, "Sun spot," 2007, `http://www.sunspotworld.com/`.