Towards secure mobile P2P applications using JXME

Marc Domingo-Prieto
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Carrer Roc Boronat, 117
08018 Barcelona, Spain
mdomingopr@uoc.edu

Joan Arnedo-Moreno
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Carrer Roc Boronat, 117
08018 Barcelona, Spain
jarnedo@uoc.edu

Jordi Herrera-Joancomartí
Escola Tècnica Superior d'Enginyeria
Universitat Autónoma de Barcelona
Campus de Bellaterra, Spain
jherrera@deic.uab.cat

Josep Prieto-Blázquez
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
Rambla del Poblenou, 156
08018, Barcelona, Spain
jprieto@uoc.edu

Abstract

Mobile devices have become ubiquitous, allowing the integration of new information from a large range of devices. However, the development of new applications requires a powerful framework which simplifies their construction. JXME is the JXTA 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 analyzes the current state of security in JXME and proposes a simple security mechanism in order to protect JXME applications against a broad range of vulnerabilities.

1 Introduction

Peer-to-peer (P2P) applications have become highly popular 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 [10], to more complex ones such as GIS (Geographic Information Systems) or e-learning [33, 17]. However, currently, Internet has become witness to the transition from a desktop-centric environment towards one based on the ubiquity of mobile devices [26]. Therefore, it was natural that the next step in the evolution of P2P applications would be following this trend [27], 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 [15, 8], among which JXME [29] can be found, the mobile version of the well known JXTA platform [28]. 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 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.

Our purpose in this paper is twofold. First of all, we examine the current state of JXME security, focusing in one of the two existing versions, JXME-Proxied. This study analyses basic peer operations by taking into account the whole peer life cycle, instead of in an isolated manner. From this study, it is possible to identify the available security mechanisms and how they operate, which may prove useful to application developers. Once the current vulnerabilities have been identified, we propose JXME-PLAuth, a simple protocol that bypass some of JXME's security shortcomings while still taking into account that it will be executed on limited devices.

The paper is organized as follows. Section 2, provides a brief overview of the JXME architecture. In Section 3, we present a security analysis of JXME from a typical peer operation cycle standpoint. Once current security has been assessed, Section 4 describes our proposal, JXME-PLAuth. The protocol security is evaluated in Section 5 and Section 6 presents the experimental results on regards to its performance on mobile devices. Finally, Section 7 summarizes the paper's main contributions and outlines further work.

2 JXME overview

As the response from the JXTA developer community to accelerate development in the wireless applications over mobile devices, Sun Microsystems presented a version of JXTA for Java 2 Micro Edition (J2ME), called JXME, providing JXTA capabilities to mobiles devices [4]. In fact, two distinct versions were developed, in order to accommodate to a broad set of scenarios. On one hand, the *JXME-Proxied* version, with very limied devices in mind, which delegates all heavyweight work to a JXTA super-peer. On the other hand, the *JXME-Proxyless* version is a straightforward port of JXTA, where peers may directly interact with the JXTA network. In this paper, we will mainly focus on the Proxied version, being the one which actually takes into account mobile device limitations on its protocols and diverging from the basic JXTA architecture, thus needing special consideration.

JXME is clearly based on JXTA, since they share the same basic specification. A detailed explanation of JXTA's generic protocols and services can be found in [30, 20] and in several papers in the literature, such as [4, 14, 24, 31]. In both JXTA and JXME, the basic organizational foundation is the *Peer Group*, a set of peers with common interests which agree on common services. Peer Groups are managed by the *Membership Service*, one of JXTA's core services, which manages the group members' identities within the group context. Identities are assigned by successfully completing an authentication process prior to actually joining the group. The Membership Service is defined as generic in the JXTA specification, leaving it up to developers to implement their own version, with the security level required by their applications.

Once a peer has joined a Peer Group, any resource may be shared with other group members by distributing its associated *Advertisement*, an XML metadata document describing the resource properties and how it may be accessed. A network resource cannot be accessed without previously recovering its associated Advertisement. Advertisements are located and distributed using the *Discovery Service*. Every time an Advertisement is retrieved by a peer, it is stored in a local cache and assigned an expiration date. On that date, the Advertisement will be automatically flushed. Once a resource has been successfully located, messaging may begin using JXTA *pipes*, abstract endpoints which provide an asynchronous unidirectional communication channel.

Figure 1 shows the main components of the JXME architecture and how their devices are integrated into a JXTA network. Devices using the Proxyless version may directly communicate with JXTA peers, whereas those which use the Proxied version version are named *Proxied Peers* and, since they are assumed to have very limited resources, cannot directly communicate with other peers.

All communications from a Proxied Peer are actually destined to a super-peer which will overcome

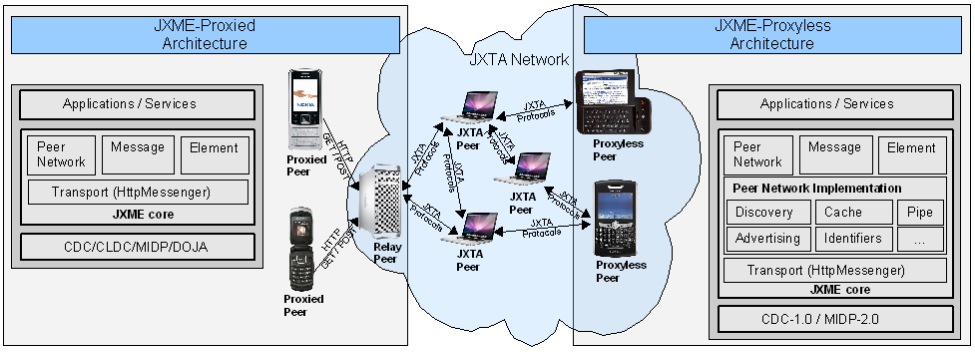

Figure 1: JXME-Proxied architecture

its limited capabilities, called *Relay Peer*, which implements the Relay and Proxy JXTA services. The Relay Peer assigns JXTA PeerID to the Proxied Peer before it may interact with the JXTA network. Furthermore, it translates or summarizes requests and responds to queries on its behalf.

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 with regards to its Proxied Peers are to listen to and answer requests from them, translate messages received to XML and retransmit them to the JXTA network, store messages received from the JXTA network for Proxied Peers and summarize and translate XML messages from the JXTA network into a simple format which Proxied Peers are able to understand.

Due to its limitations and reliance on a Relay Peer, the kind of operations that a Proxied Peer can actually execute are limited to a very small set. First of all, Proxied Peers may **join** a group. Once the peer has successfully performed this operation, it may interact with other group members by **searching** or **creating** resources (such as Peer Groups or pipes), **listening** to a pipe to receive data, **sending** data to a specific pipe, **closing** a pipe and **polling** the Relay Peer for messages from the JXTA network that have the Proxied Peer as the final destination.

All message exchanges between a Proxied and Relay Peer share a special simple protocol encapsulated using HTTP-POST. To reduce the number of messages sent to the Relay Peer, they are stored in a queue at the Proxied Peer and each time a poll operation is performed, the first message in the queue is actually sent to the Relay Peer.

In the communication between a Proxied and Relay Peer each message starts with a special header where all namespaces are declared, and then is structured as a list of individual elements which contain the request type and its associated parameters. Every element is formatted as simple text, in contrast with standard JXTA, which uses XML, and always follows the same syntax. The element fields have the following order and functionality:

- Starting, a `jxel` string.
- The element's namespace identifier, chosen from the ones declared in the message header.
- An optional flag specifying additional element properties, such as content encoding.
- The element type, which is composed by two sub-fields: the type string length and the type itself.
- The element content, also composed by two sub-fields: the content string length and the type itself.

A sample message, is shown in Figure 2. All elements would actually be sent in a single line, but the header and each element have been put in different lines and the text has been formatted for the sake of

readability. In this example, for instance, in the first element (second line), the namespace with id 2 is used, the element type is `request` (a string 7 characters long), and the content is `search` (6 characters long). This means that a `search` operation is being requested to the Relay Peer. The other elements are the parameters associated to such kind of request, and will vary depending on the request type.

Listing 1 - JXME-Proxied sample message

```
jxmg 0 01 05 proxy     08
jxel 2  0 07 request   0006 search
jxel 2  0 04 type      0005 GROUP
jxel 2  0 04 attr      0004 name
jxel 2  0 05 value     0006 myName
jxel 2  0 09 threshold 0001 2
jxel 2  0 09 requestid 0001 6
jxel 1  0 26 EndpointDestinationAddress xxxx <destination address>
jxel 1 0 21 EndpointSourceAddress       xxxx <source address>
```

Figure 2: JXME-Proxied sample message

From this overview of the JXME, focusing on the Proxied version, it is obvious that the need for 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.

2.1 Related research on JXME security

As it has been pointed out, security is a key feature in current P2P middlewares. Unfortunately, to our best knowledge, not many proposals exist in the literature for JXME.

Authors in [22] present an infrastructureless network composed of limited mobile devices called Mobile Ad hoc NETworks (MANETs). For them, JXME is a significant attempt at designing middleware for mobile devices.

On regards to the Proxyless version, since it is is a direct, though somewhat simplified, port of JXTA, supporting the same protocols and architecture, proposals that apply to JXTA also apply to this version, such as [3]. Nevertheless, an extensive survey on the state of the security in JXTA can be found here [2].

As far as the Proxied version 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 [12]. This authentication is provided by the Relay Peer, which uses an external Sign and LDAP Server, breaking completely the P2P model proposed by JXTA.

To sum up, JXME-Proxyless version maintains the same structure as JXTA and therefore can inherit its improvements, such as new security schemes. Oppositely, JXME-Proxied version uses a simplified protocol and cannot inherit directly security improvements from JXTA. And, to our best knowledge, there is only one authentication proposal that tries to increase JXME-Proxied security baseline, but provides it in a centralized manner.

3 JXME-Proxied Security analysis

A global security analysis on standard JXTA already exists in [2]. Since the Proxyless version operates just like standard JXTA, most of its conclusions apply. However, it does not apply to the Proxied version because of its divergence from the base JXTA model, by relying some operations on a Relay Peer. Therefore, we will only focus on analyzing the security degree of the Proxied version. From this study, we can identify and assess existing vulnerabilities. Nevertheless, the analysis follows the same methodology proposed in the aforementioned study, where the typical peer life cycle is examined rather than isolated peer actions.

In order to perform a security assessment, it is useful to identify and categorize the most common types of attack in P2P networks. All attacks can be divided into two distinct groups, according to the degree of involvement of the attacker [6]: passive attacks, where the attacker just monitors peer activity and network traffic, and active attacks, where the attacker purposely interferes with data or network activity. Each group can be further classified according to the particular action performed by the attacker.

Passive attacks which have been considered are *Eavesdropping* and *Traffic analysis*. The former consists of searching in message exchanges for sensitive information, such as passwords, whereas the latter analyzes traffic data looking for patterns and relevant peers.

Active attacks include *Spoofing*, *Man-in-the-middle*, *Replay*, *Local data alteration* and *Software security flaws*. Spoofing consists in impersonating another peer. Man-in-the-middle (MitM) intercepts the communications between two parties transparently relaying forged messages to each one. Replay captures messages so they can be reused at a later time to simulate a real message exchange initialization. Local data alteration modifies local data to corrupt the system behavior. Finally, Software security flaws exploit vulnerabilities due to bugs in the source code, trying to obtain unexpected actions on the software.

This assessment will focus on the communications between the Proxied and Relay Peer, also taking into account the way a Relay Peer stores and manages its subscribed Proxied Peers' data. All communications between the Relay Peer and the rest of the JXTA network operate under the standard JXTA security model, so it is out of the scope of ths paper since it has been deeply analyzed in [2].

The standard JXME peer general operation cycle can be summarized in the following stages: Platform startup, Peer Group joining, Resource discovery and publication, Message exchange and Disconnection. The security analysis follows the actions performed by a Proxied Peer according to this lifecycle.

3.1 Platform startup

The first step during platform startup is loading the JXME libraries into the system. Unfortunately, no security model has been considered in order to distinguish correct JXME binary releases from another one with malicious code (*Local data alteration*).

Once the platform binaries have been loaded, but before a Proxied Peer may join the JXTA network, it must associate with any available Relay Peer. During this process, according to the JXME specification, the Relay Peer generates a new peer identifier, *PeerId*, and sends it back to the Proxied Peer. Such an identifier is only used within the context of message exchanges between Proxied and Relay Peers and has no prevalence in the general JXTA network. At this point a Proxied Peer joins (next operation) to a default Peer Group, the NetPeerGroup, obtaining a JXTA PeerId.

The PeerId generation process is very important, since the Relay Peer is only able to identify Proxied Peers by their PeerId. However, we have 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 described in

Section 2, since the request is sent using HTTP-GET. An example of an identifier request message is shown in Figure 3. It can be recognized as such since the PeerId is specified as `unknown-unknown` in the GET command.

Listing 2 - Proxied Peer PeerId request message

```
GET /unknown-unknown?0,-1,http://192.168.0.37:2481/
EndpointService:jxta-NetGroup/uuid-DEADBEEF...0F05/pid HTTP/1.1
Connection: close Content-Length: 0
User-Agent: UNTRUSTED/1.0
Host:172.16.0.37:2481
```

Figure 3: Proxied Peer PeerId request message

Sending the PeerId requires an initial communication protocol, which makes Proxied Peers very vulnerable at startup, since the identifier is transmitted in clear text over the network, allowing an attacker to easily learn it (*Eavesdropping*). Furthermore, since no authentication exists between a Proxied and Relay Peer, an attacker can also act as an invisible intermediate with the Relay Peer, redirecting HTTP messages (*Man-in-the-Middle*). At this stage, reusing intercepted data (*Replay*) makes no sense, since each peer starts the platform only once.

Even though JXME does not take into consideration a secure startup stage, an initial authentication protocol based on PKI is described in the Trusted Group proposal [14], out of the scope of the JXME's specification. The mechanism provides unidirectional authentication, only the Proxied Peer authenticates the Relay Peer, ensuring it is a legitimate one. However, this authentication is provided using a central Sign Server and LDAP database, which partially breaks the P2P model.

3.2 Peer Group joining

A Proxied Peer may join any Peer Group through the Relay Peer. As a requisite to proceed with this process, all group members must agree to use the same Membership Service implementation. This is achieved by sending a message with a `request` element which contains the `join` string. Group Membership parameters are included in the rest of the message elements.

The default Membership Service implementation in JXME-Proxied is the *None Membership Service*, which is used in groups without any kind of authentication, where any peer may claim any identity. It was designed for applications with no security requirements. As a result, all data exchanges based on this Membership Service are completely insecure.

Since the information in transmitted messages is sent in clear text, an attacker may discover the group any Proxied Peer is trying to join (*Eavesdropping*) and identify important peers by its traffic (*Traffic analysis*). Also, an attacker may easily impersonate any peer by claiming the other peer's identity within the Peer Group (*Spoofing*). However, during the join operation, reusing a directly captured message (*Replay*) is pointless since a Proxied Peer can only join once to a Peer Group during a single session. Therefore, replay attacks are not a real concern.

In order to address some of these vulnerabilities, the proposal in [14] also extends its basic principle to provide an additional implementation of a Membership Service, external to the JXTA specification. An authentication mechanism is provided between the Proxied Peer and the Peer Group. However, in this case, it is based on mutual authentication (the Peer Group itself may also be authenticated). This authentication is based on certificates from both the Proxied Peer and the Peer Group and an external Sign and LDAP server.

3.3 Resource discovery and publication

Resource publication and discovery are also fully managed by the Relay Peer. The Proxied Peer just sends requests, using the HTTP-POST protocol, asking for Advertisements to be created or located.

Unfortunately, as a result of sharing the same simple protocol, an attacker can easily publish false resources (*Spoofing*) and modify/delete the Advertisements (*Man-in-the-middle*) of any other Proxied Peer, since no authentication is enforced. Any peer may create Advertisements on any other peer's behalf at leisure. Furthermore, an attacker can resend captured messages performing the original operation several times (*Replay*) in order to produce multiple resource discovery queries.

Finally, Advertisements are transmitted without encryption and can be easily intercepted (*Eavesdropping*) by an attacker, which can recognize important peers, those sharing many resources, by analyzing its traffic (*Traffic analysis*).

3.4 Message exchange

In JXTA, network messages are exchanged using pipes, briefly introduced in Section 2. Unfortunately, Proxied Peers are not able to use pipes between them and the Relay Peers. Since pipe usage is a complex mechanism which requires a non-negligible amount of system resources, it is the Relay Peer which, again, actually manages pipes, connecting to services in the JXTA network on behalf of the Proxied Peer. The communication between a Proxied and Relay Peer is performed using HTTP.

Pipe management requests are also based on a generic `request` element type. The element content dictates the actual operation: `create`, to create the pipe, `listen`, to receive messages, and `send`, to send messages. There is no specific element type to receive pipe messages. They are automatically transmitted from the Relay Peer each time any request is received from the Proxied Peer, along with the request reply.

As a result, different attacks can be performed: *Eavesdropping*, *Traffic analysis*, *Spoofing*, *Man-in-the-middle* and *Replay*. These attacks mainly allow an attacker to send/receive and sniff messages, as well as impersonate any peer. Moreover, an attacker can close legitimate peer pipes at will, abruptly ending message exchanges.

3.5 Disconnection

Before a peer may disconnect from the JXTA network, all pipes should be previously closed. However, the main limitation at this step is that no operation currently exists in JXME-Proxied for this purpose. It is also the Relay Peer that has to decide when to unsubscribe a peer from a Peer Group. Therefore the victim's PeerId may be easily spoofed, stealing his open pipes, opening new pipes, preventing pipe disconnection, and using all the groups previously joined.

3.6 JXME-Proxied security evaluation summary

Since JXME-Proxied is Open Source Software (OSS), supported by a community of developers, and it is also very simple and small, it could be considered relatively safe from *Software security flaws*. Furthermore, since Proxied Peers do not store data locally, they are not vulnerable at execution time to *Local data alteration*.

The analysis of possible attacks and the existing security mechanisms of JXME-Proxied, classified by peer operations, provides a vulnerability map summarized in Table 1.

The four main vulnerabilities found are:

- **V(1)**: malicious executable code can easily be built and cannot be automatically discovered when installed

Op./Threat	Evs	TAn	Spf	MitM	Rp	LDA	SSF
Startup	V(2)	N/A	V(4) P(TGMS)	V(2, 4) P(TGMS)	N/A	V(1)	P(OSS)
Join	V(2)	V(3)	V(4) P(TGMS)	V(2, 4) P(TGMS)	N/A	N/A	P(OSS)
Publish/ Discover	V(2)	V(3)	V(4)	V(2, 4)	V(4)	N/A	P(OSS)
Messaging	V(2)	V(3)	V(4)	V(2, 4)	V(4)	N/A	P(OSS)
Disconnect	V(2)	V(3)	V(4)	V(2, 4)	N/A	N/A	P(OSS)

N/A: Non-applicable.
V(type): Vulnerability exists.
P(mechanism): Security mechanism used

Table 1: JXME-Proxied peer operation cycle security summary

- **V(2)**: no encryption mechanism exists
- **V(3)**: no data flow masquerading mechanism exists
- **V(4)**: no actual authentication is enforced

The available security mechanisms are:

- **P(OSS)**: Open Source Software
- **P(TGMS)**: Trusted Group Membership Service [14]

4 A secure protocol extension for JXME-Proxied: JXME-PLAuth

The security analysis presented in Section 3 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 this paper, we present JXME-PLAuth (JXME-Proxied Light Authentication), a proposal to avoid *Spoofing* and *Replay* attacks. This proposal does not try to solve every single vulnerability which was identified, which would require a much broader set of security mechanisms, but provides a initial protection in the communication by guaranteeing lightweight authentication. This is the first step in providing a secure mobile framework for JXME.

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 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 extension is based on lightweight cryptographic operations, mainly hash functions. This approach has been successfully used in other proposals for MANET based environments, such as [13].

Protection against Spoofing and Replay attacks may be obtained by securely identifying the Proxied Peer using any well known authentication scheme. Our proposal relies on a lightweight scheme, 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 the use of a hash-chain [16] based scheme to create a set of linked values which will be used as local identifiers.

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.

Instead of obtaining a PeerId from the Relay Peer, Proxied Peers generate themselves 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.

4.1 Protocol initialization

The proposed scheme needs an initialization process executed during the Proxied Peer's startup step and boot operation. In this process, the hash-chain is created, values are stored in the Proxied Peer's internal memory and the first PeerId is transmitted to the Relay Peer.

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 4 summarizes this process.

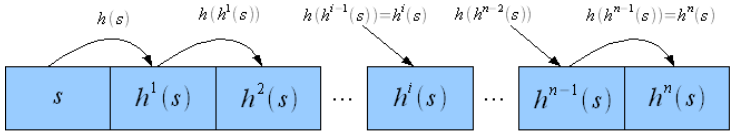

Figure 4: Hash-Chain creation

3. The Proxied Peer's initial PeerId, $initId$, is created from $h^n(s)$, fulfilling the JXTA peer identifier specification. A PeerId may be created from any value in $h^n(s)$ according to the following steps (summarized in Figure 5):

   - The PeerId starts with the string $uuid-$.
   - The 16 most significant bytes, $\text{msb}_{16}(initId)$, 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 insecure protocol, $initId$ is announced.

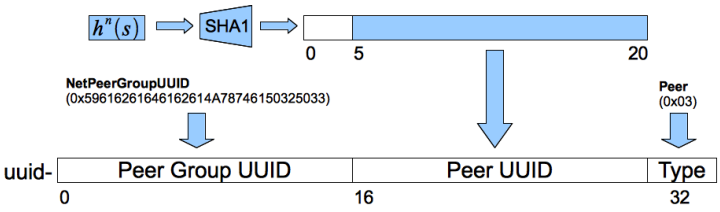

Figure 5: initId generation

5. When the request is received, the Relay Peer randomly generates a new JXTA PeerId, *jxtaID*1, 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 2) 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` = *jxtaID1*, where *initId* is considered the entry's key.

4.2 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:

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

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

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

The identifier consumption from $hc(s)$ and the translation between the changing PeerId and the static JXTA PeerId is represented in Figure 6. 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 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 follows:

1. The Relay Peer takes the message identifier $currentId$.

2. The PUUID section of the identifier, $currentPUUID$, is extracted from $currentId$.

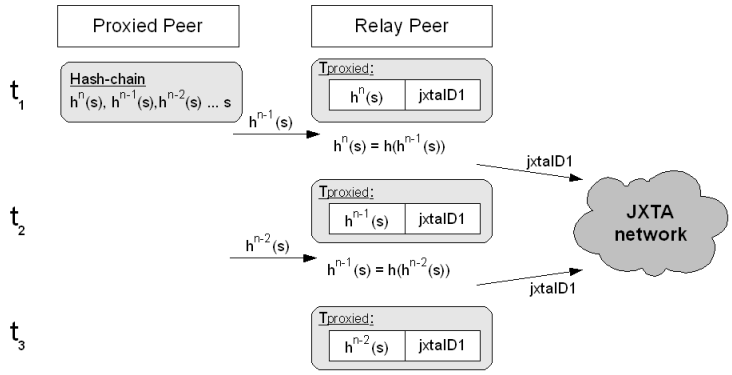

Figure 6: Hash-Chain consumption and identifier translation

3. If *currentPUUID* matches with $T_{proxied}$'s `lastId` field, it means that someone is trying to perform a spoofing or replay attack. Therefore, this petition is obviated. Otherwise, the authentication protocol continues.

4. The Relay Peer calculates $h(currentPUUID)$. From this value a PeerId is generated following the steps described in Figure 5.

5. The result is looked up among the currently stored values in $T_{proxied}$'s `lastId` field.

6. 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.

7. The Relay Peer stores *currentId* into $T_{proxied}$ replacing the old value matched in step 5. It becomes the entry's new key.

8. 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.

4.3 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 as its associated parameter the new Id:

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

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.

As it has been mentioned before, the proposed approach avoids *Spoofing* and *Replay* attacks. Furthermore, this 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 `listen` request, just like 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.

On regards to the computational cost of this proposal, it is worth mention that the only cryptographic operations used are hash value computations. Hash values are lightweight cryptographic operations which can be efficiently computed even in constrained devices.

5 JXME-PLAuth security evaluation

JXME-PLAuth adds simple mechanisms to provide protection against Spoofing and Reply attacks. Other attacks, such as Man-in-the-middle attacks, are not protected. This is because our proposal tries to be as simple as possible. Also, these attacks are not expected in some scenarios, such as when communications are direct.

The mechanism used to renew the hash-chain is authenticated using the same mechanism of hash-chain identifiers, therefore it is as secure as normal authentication using hash-chains.

Some messages can be lost, this can be solved in several ways [23, 7]. An easy integration of this mechanism in JXME-PLAuth can be done by adding a new parameter in exchanged messages containing the position of the peer identifier in its hash-chain. Also, the Relay Peer has to store and maintain updated this new information in its local translation table. Then, when the Relay Peer receives a message which contains a peer identifier that does not match in the local translation table, it can perform further hashes based on the identifier and the position in the hash-chain. If a matching is produced means that a previous message has been lost, but by this mechanism the local table is updated and further messages of this Proxied Peer can still be secure authenticated.

A security analysis similar as the one performed in Section 3 is conducted to our security proposal, JXME-PLAuth. This analysis follows the same methodology as before and just the security in the communications between the Proxied and the Relay Peer are analyzed.

5.1 Platform startup

In this stage the Proxied Peer becomes linked with its Relay Peer. Now it is safe of Spoofing attacks, since no other peer can authenticate itself impersonating another peer.

If one Peer tries to do so, it is going to receive an error. This is explained in Section 4.2. Peers can know current identifiers of other peers but cannot predict their next identifiers and therefore peers are secure authenticated.

There are two possible scenarios potentially prone to suffer security threads:

1. Two Proxied Peers use the same identifier but in different Relay Peers.

2. A Proxied Peer (P1) is authenticated in a Relay Peer (R1). Another Proxied Peer (P2) capture P1's identities and authenticate itself in R1 but always using the previous identifier of P1.

Both scenarios are extreme but possible cases. To understand how this protocol deal with these scenarios is important to have in mind Figure 6, where is shown that there are two identifiers: One internal identifier between the Proxied Peer and the Relay Peer, and one global identifier inside the JXTA Peer Group. This analysis, and in general all the paper, is only focused on the first identifier, the internal one.

In the first scenario both Proxied Peers will have the same internal identifier but in different Proxied Peers. And these peers will probably have different JXTA identifiers, but this is responsibility of the JXTA Membership Service, outside of the scope of this analysis. Therefore, secure authentication is provided.

In the second scenario both Proxied Peers (P1 and P2) will be authenticated at the same Relay Peer. At any moment, P1 will have its identity based on $h^i(s)$ while P2 in $h^{i+1}(s)$ (previous P1's identifier). Therefore, both internal identifiers are different. P2's identifier must always come, at minimum, one step before P1's identifier, otherwise P2 is going to receive an error when authenticating. Also, as previous scenario, the JXTA identifier of both peers should be different.

5.2 Peer Group joining

A Proxied Peer has to send a message to its Relay Peer in order to request to join a Peer Group. In this case, this operation is protected against Spoofing attacks since only the Proxied Peer can know the next identifier in the hash-chain.

5.3 Resource discovery and publication

This operation is also performed by a Proxied Peer by sending a message to its Relay Peer. In this case the Proxied Peer is also secured from spoofing attacks since non other peer different than it can send a message with its next identifier to its Relay Peer to find or create a resource in its behalf. Also, the message is protected against Replay attacks since when a message is resend with the same identifier, the Relay Peer will discard it.

5.4 Message exchange

Performing this operation produces a similar message as the previous step. In this case, Spoofing and Replay attacks are as well secured since no one knows Proxied Peer's next identifier and captured packages will be discarded by the Relay Peer if replayed.

5.5 Disconnection

In this operation, when a Proxied Peer decides to leave the network, it still does not have a specific operation to remove completely all its information stored in its Relay Peer. But using this authentication scheme prevents that other peers could perform Spoofing attacks, like closing its open pipes. Also, this mechanism guarantees that when a Proxied Peer disconnects from the network no one can steal its information contained in its Relay Peer. This is because the authentication scheme will remain working until the Relay Peer decides to remove Proxied Peer's data.

5.6 JXME-PLAuth security summary

The security analysis presented in this section verify that our security proposal JXME-PLAuth provides an enough strong authentication mechanism that prevents Proxied Peers from Spoofing and Replay attacks. From Table 1,

Table 2 has been generated in order to summarize this analysis. It shows that the vulnerability produced by the lack of an authentication scheme (V(4)) has been patched. Therefore, those attacks that were allowed only due to this vulnerability are avoided.

Op./Threat	Evs	TAn	Spf	MitM	Rp	LDA	SSF
Startup	V(2)	N/A	**P(JXME -PLAuth)** P(TGMS)	V(2) P(TGMS)	N/A	V(1)	P(OSS)
Join	V(2)	V(3)	**P(JXME -PLAuth)** P(TGMS)	V(2) P(TGMS)	N/A	N/A	P(OSS)
Publish/ Discover	V(2)	V(3)	**P(JXME -PLAuth)**	V(2)	**P(JXME -PLAuth)**	N/A	P(OSS)
Messaging	V(2)	V(3)	**P(JXME -PLAuth)**	V(2)	**P(JXME -PLAuth)**	N/A	P(OSS)
Disconnect	V(2)	V(3)	**P(JXME -PLAuth)**	V(2)	N/A	N/A	P(OSS)

N/A: Non-applicable.
V(type): Vulnerability exists.
P(mechanism): Security mechanism used

Table 2: JXME-Proxied using JXME-PLAuth extension peer operation cycle security summary

6 JXME-PLAuth experimental results

Some real experiments were done to test how the proposed security improvement to the JXME-Proxied framework impacts its overall behavior. The actual performance of our proposal implementation has been evaluated by assessing how the protocol extensions in 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.

The mobile device needs network connectivity to exchange data with the Relay Peer and an open operating system which allows the execution of J2ME applications and obtaining information about the state of its resources, such as memory or battery usage. There are many operating systems for mobile devices, such as iOS [1], WebOS [21], Windows Phone 7 [19] or Android OS [11], but the one that fulfilled most our requirements was the Android OS. The main reason is the fact that this mobile operating system is based in a modified version of the linux kernel, which allows a high degree of customization and direct access to device resources. In addition, Android OS is becoming very popular and is being used in a large range of devices. Finally, although it does not allow to natively run J2ME applications, it natively supports Java. But, since native Android OS applications are being more popular than J2ME ones, it motivates us in spending some effort in customizing JXME-Proxied to be able to execute it in the large amount of devices that run Android OS.

The mobile phone chosen was the HTC Hero. This device was one of the most popular Android mobile phones at the moment of selecting a device for our tests but compared with new mobile phones its hardware is obsolete. Therefore, if JXME-PLAuth can be handled by this device, new devices will deal with it better. The HTC Hero has a processor Qualcomm MSM7200A 528 MHz, 512 MB of ROM, 288 MB of RAM, display of 3.2-inch TFT-LCD touch-sensitive with 320x480 HVGA resolution, Wi-Fi IEEE 802.11 b/g and rechargeable Lithium-ion battery with 3.7 V and 1350 mAh of capacity.

On regards to the computer, it has to be powerful enough to act as a Relay Peer and addressable, to receive communications from the Proxied Peer. Most of today's computers meet these requirements. The chosen computer has been a laptop computer, a MacBook Pro, which can be addressed through Wifi or Ethernet.

The communication channel used was Wi-Fi, since it is the one shared between both devices, and a wireless router was used in order to connect them.

The tests have to provide an idea of the overhead produced in a Proxied Peer implementing security. It

is also assumed that a minimum overhead will be produced in the Relay Peer, but since it is expected that it has enough capabilities to deal with the extra work, these tests are not going to focus on it. The main test consisted in assess the behavior of a real JXME-Proxied application, a chat application, comparing a version which implements security with one which does not.

6.1 Test application

This test assessed the impact of the proposed security mechanism in a real application. The goal of this test is to detect the general influence of using security in the JXME messaging service.

The default JXME-Proxied distribution comes with a couple of sample applications to show the potential of this framework. One of these applications is a simple chat, which allows Proxied Peers to exchange messages between them.

This chat was chosen as the application for this test for many reasons:

- Message exchange is a typical operation in mobile devices. Until recently, this operation has been done using Short Message Service (SMS).
- With the extension of having access to the Internet network from mobile devices, some chat applications [32, 18] are getting really popular in this environment.
- This application is focused in communication rather than in computation, which allows an intensive test of JXME.
- The core structure of a chat application is simple.

However, since this chat is written in J2ME language and the Android OS cannot run it directly, some modifications have been performed. This is because while Android applications are written in Java, there is no Java Virtual Machine in the platform and Java bytecode cannot be executed. Java classes get recompiled into a Dalvik executable and run on a Dalvik virtual machine. Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU. The modifications done in this chat application are basically the graphical user interface (GUI) and the J2ME framework specification used in the class responsible of the communication, the `HttpMessenger` Java class.

JXME-Proxied supports both CLDC and CDC J2ME frameworks specifications. The use of a different framework specification only affects in the HttpMessenger class, which is the one responsible for the communication. By default, JXME-Proxied uses CLDC, but Android does not support the classes used. Fortunately, all the classes used in CDC are supported by Android. Therefore, in our chat application, we have chosen the CDC version of the HttpMessenger class. Theoretically, all versions of this class should have worked properly, but the tests performed showed that the CDC version was not working correctly. However, after some small changes (always using the CLDC version as a model) the behavior of this class was fixed.

In this test the measured and compared resource was the battery consumption. Figure 7 shows some snapshots of the chat application running in Android.

6.1.1 Experimental results

This test measures the impact of JXME-PLAuth in a real application. Some considerations were taken:

- Screen light: Set to the minimum, since it is enough in most of the environments.

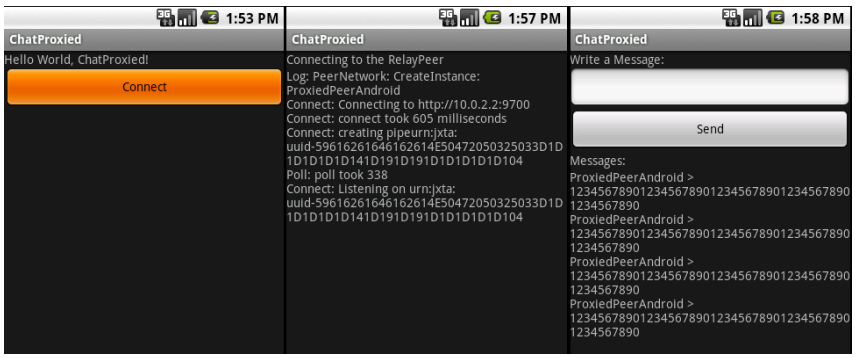

Figure 7: Chat Proxied GUI

- Message exchange rate: A chat application is used in this test and we want to run it according to an average usage. This data is extracted from [5], where an analysis of the typical characteristics in instant messaging is conducted. The conclusion is that, typically, a message is sent each 10 seconds and the average size of each message is 32 chars.

- Hash function: SHA1 [9] has been used as the hash function, but any other hash function can be used.

The main test compares the application with and without security. In both cases the time interval between polls (operation performed by the Proxied Peer to send and receive messages to/from the Relay Peer) has to be defined. When security is activated, the size of the hash-chain also has to be specified.

Based on the poll time used in sample applications, and the time it takes to compute hash-chains of different sizes , the poll interval values that are considered significant are 1, 5 and 10 seconds (15 seconds are not considered since in this application the time between sent messages sent is 10 seconds). Different hash-chain sizes can be used for each test, but a logical restriction has been applied: The maximum time required to compute the hash-chain must be smaller to the time between polls. Table 3 shows the different possible combinations.

In Figure 8 the accumulated energy consumption of the chat application when using security and different time poll intervals and hash-chain sizes is presented (just trend lines are shown for the sake of readability). This shows that, in general, increasing poll time intervals and hash-chain sizes decreases the consumption of the battery. Having a constant hash-chain size and increasing the poll time interval always decrease the consumption. If the hash-chain size is 50 the percentage decrease of consumption for 1 and 5 seconds of poll interval is 10,77%, and for 5 and 10 seconds is 55,54%. Whereas when using a hash-chain of 250 elements the percentage decrease of consumption is 84,28% for 5 and 10 seconds. But having a constant poll time and increasing the hash-chain size not always decreases the battery consumption. If we use a poll interval of 5 second, the percentage decrease of consumption is 1,14% for hash-chains of 50 and 250 elements (this small difference between these two scenarios causes

	HC size		
Poll interval (s)	50	250	500
1	✓	×	×
5	✓	✓	×
10	✓	✓	✓

Table 3: Combination of values of time between poll intervals and size of hash-chain used for the tests

that in the Figure 8 both green lines are overlapped). And if 10 seconds is used, the percentage decrease of consumption is 17,15% for hash-chains of 50 and 250 elements, and -49,81% for hash-chains of 250 and 500 elements. The increase of consumption in this last case we think that is caused because the hash-chain is too big and does not fit into the L1 cache of the mobile phone (32 KB) and therefore more energy and time is required to calculate and consume the hash-chain. These values indicate that increasing the poll time intervals has higher impact in decreasing the consumption than increasing the hash-chain size.

It shows that renewing less and computing a longer hash-chain is more efficient than doing it more frequently.

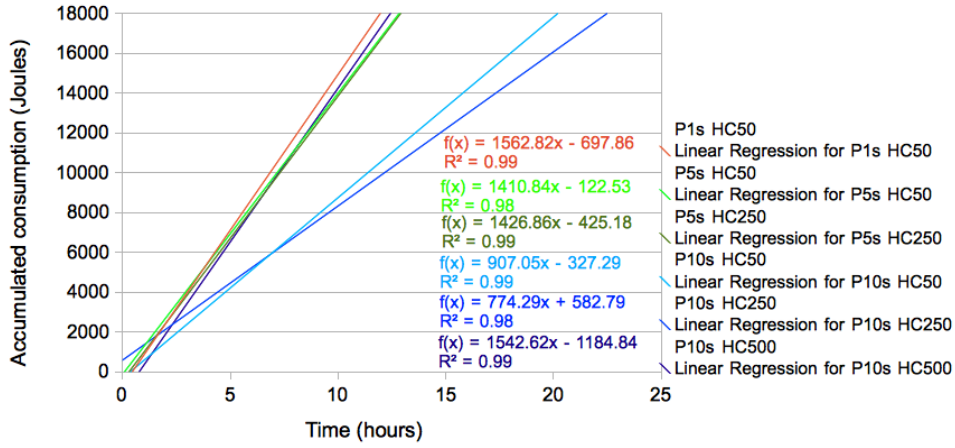

Figure 8: Accumulated consumption when running the chat application with security and different poll intervals and hash-chain sizes

To provide some perspective on energy consumption, the device's battery capacity is 1350mAh and its voltage is 3.7V. Just multiplying both values (converting the hours to seconds) we get that the battery has about 17982 Joules. When the device is functioning at maximum bright screen, the consumption is 22,5 J/min, which amounts to 6750 Joules/5hours, 13500 J/10h or 20250 J/15h. Therefore, even in the most intensive scenario (P1s HC50), the consumption is very similar to having the device idle with the maximum bright screen.

Figure 9 compares the accumulated battery consumption between using security or not in JXME-Proxied when running the chat application at different poll time intervals. When security is used, the hash-chain size which produces the better result in terms of battery usage is chosen for the different poll time intervals. This figure shows that in general the overhead produced by utilizing security is low. This overhead was expected because some security measures were used, but this impact also depends on the poll time. The percentage increase of consumption when using security and 1, 5 or 10 seconds as poll time interval is respectively 0,86, 48,77 and 2,3. And in the worst case, 5 seconds, the overhead produced can be reduced by playing with the size of the hash-chain.

6.2 Evaluating the performance compared to HTTPS

All previous performance experiments have been done over our light authentication proposal, JXME-PLAuth. But is also important to compare the obtained results with the overhead produced with one of the typical authentication scheme, such is HTTPS [25]. Fist of all, It has to be pointed out that HTTPS additionally of guaranteeing authentication also provides encryption. But a trusted entity is required to manage certificates to guarantee this security level.

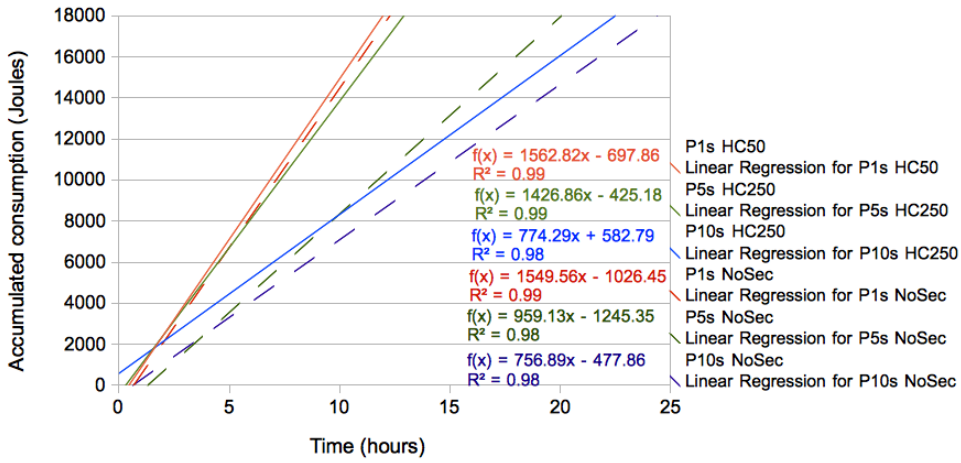

Figure 9: Accumulated consumption when running the chat application with security (continuous line) and without security (dashed line)

A experiment was conducted in order to compare the behavior of the system when using HTTPS instead of simple HTTP in the same scenario used in Section 6.1. The results showed that while the battery consumption of using HTTPS instead of HTTP is quite similar, the time required to perform HTTPS is enormous. Establishing connection and sending data in HTTP in average only took 0.5 seconds, while in HTTPS it took 26 seconds.

6.3 Tests conclusions

Previous tests analyze the impact caused by using security in mobile devices. These tests show that in general the overhead produced by JXME-PLAuth is low and acceptable. This overhead depends basically on the time between polls and the size of the hash-chain used. Increasing the time between polls always reduce the overhead. However, increasing the size of the hash-chain does not always reduce the overhead. For this reason, is important to chose a long hash-chain in order to reduce the times the hash-chain has to be renewed, but not too long that modify memory access patterns. Using higher authentication schemes, such as HTTPS, produces an important overhead, which justify the usage of simpler authentication schemes.

7 Conclusions and Future Work

JXME is the JXTA simplification that allows to run P2P applications using the JXTA milddleware when some peers are mobile devices with computational constraints. As we have analyzed in this paper, such simplification takes an extreme effect in the JXME-Proxied version, where the mobile device constraints imply that some of the tasks cannot be performed by the platform installed in the mobile peer and should be delegated to a Relay Peer with more computational resources.

From the security point of view, we have performed an analysis that shows the relevance of the link between the Proxied Peer and its Relay Peer. Without properly securing such link, a wide range of attacks can be performed, from passive attacks, like Eavesdropping or Traffic analysis to more active ones such as Spoofing, Replay or Man-in-the middle.

The security analysis performed has allowed us to understand the nature of such vulnerabilities and then we have been able to provide an effective mechanism that allows to protect the communication be-

tween the Proxied Peer and its Relay Peer from Spoofing and Replay attacks. The obtained protection has been achieved using the concept of hash-chain in order to offer linkability between different communication sessions performed between the Proxied Peer and the Relay Peer. Using the onewayness of hash functions, linkability can be obtained at a low computational cost.

Such computational saving is translated in an affordable energy consumption, a really important feature when dealing with mobile devices. As it has been shown in the presented experimental results, in a standard mobile application like a messaging service running over an Android OS mobile device the inclusion of the proposed security mechanism does not impact significantly on the energy consumption of the device. Furthermore, the proposed solution does not imply to define new message protocols in JXME, since it can be implemented using standard JXME messages.

As we have discussed, other attacks rather than Spoofing and Replay can be performed on a JXME-Proxied implementation, and for that reason further research has to be performed in order to achieve the desired protection, a difficult task since the obtained solutions must be consistent with the constrained computational environment of a Proxied Peer device.

Acknowledgments

This work is partially supported by the Spanish Ministry of Science and Innovation and the FEDER funds under the grants TSI2007-65406-C03-03 E-AEGIS, CONSOLIDER CSD2007-00004 ARES and TIN2010-15764 N-KHRONOUS.

References

[1] Apple Inc. iOS. `http://www.apple.com/ios/`, 2008.

[2] J. Arnedo-Moreno and J. Herrera-Joancomartí. A survey on security in JXTA applications. *Journal of Systems and Software*, 82(9):1513 – 1525, 2009.

[3] J. Arnedo-Moreno and J. Herrera-Joancomartí. JXTA resource access control by means of advertisement encryption. *Future Generation Computer Systems*, 26(1):21 – 28, 2010.

[4] A. Arora, C. Haywood, and K. Pabla. JXTA for J2ME, extending the reach of wireless with JXTA technology. Technical report, SUN Microsystems, Inc, 2002.

[5] D. Avrahami and S. E. Hudson. Communication characteristics of instant messaging: effects and predictions of interpersonal relationships. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 505–514, New York, NY, USA, 2006. ACM.

[6] D. Brookshier, D. Govoni, N. Krishnan, and J.C. Soto. *JXTA: Java P2P Programming - Chapter 8: JXTA and Security*. Sams, 2002.

[7] Cristina Cano, Manel Guerrero, and Boris Bellalta. Secure and efficient data collection in sensor networks. In *Wireless Systems and Mobility in Next Generation Internet*, volume 5122 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin / Heidelberg, 2008.

[8] B.G. Christensen. Experiences developing mobile P2P applications with lightpeers. *Peer-to-Peer Computing, IEEE International Conference on*, 0:229–230, 2006.

[9] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). `http://www.ietf.org/rfc/rfc3174.txt`, 2001.

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

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

[12] R. Housley, W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure. `http://www.ietf.org/rfc/rfc2459.txt`, 1999.

[13] H. Janzadeh, K. Fayazbakhsh, M. Dehghan, and M.S. Fallah. A secure credit-based cooperation stimulating mechanism for MANETs using hash chains. *Future Generation Computer Systems*, 25(8):21 – 28, 2009.

[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*, volume 4, pages 116–121, Aug. 2005.

[15] G. Kortuem. Proem: a middleware platform for mobile peer-to-peer computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):62–64, 2002.

[16] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[17] 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*, 4(3):352–371, 2008.

[18] Meebo Inc. Meebo. `http://www.meebo.com/android/`, 2008.

[19] Microsoft. Windows Phone 7. `http://www.microsoft.com/windowsphone/`, 2010.

[20] Oracle. JXTA java standard edition v2.5: Programmers guide. `https://jxta-guide.dev.java.net/`, 2007.

[21] Palm Inc. WebOS. `http://developer.palm.com/`, 2009.

[22] G. Paroux, I. Demeure, and D. Baruch. A survey of middleware for mobile ad hoc networks. In *Technical Report 2007/D004*. Ecole Nationale Supérieure des Télécommunications, 2007.

[23] Adrian Perrig, Ran Canetti, and Ibm T. J. Watson. Efficient authentication and signing of multicast streams over lossy channels. In *In IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[24] T. Piedrahita and E. Montoya. Performance analysis of JXTA/JXME applications in hybrid fixed/mobile environments. *Revista Colombiana De Computación*, 7(1):5, 2006.

[25] E. Rescorla. HTTP Over TLS. `http://tools.ietf.org/html/rfc2818`, 2000.

[26] Lidan Shou, Xiaolong Zhang, Ping Wang, Gang Chen, and Jinxiang Dong. Supporting multi-dimensional queries in mobile P2P network. *Information Sciences*, 181(13):2841 – 2857, 2011. Including Special Section on Databases and Software Engineering.

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

[28] Sun Microsystems. Project JXTA. `http://www.jxta.org`, 2001.

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

[30] Sun Microsystems. JXTA v2.0 protocols specification. `https://jxta-spec.dev.java.net/nonav/JXTAProtocols.html`, 2007.

[31] T. Tahsin, L. Choudhury, and L. Rahman. Peer-to-Peer mobile applications using JXTA/JXME. In *11th International Conference on Computer and Information Technology*, volume 1, pages 702–707, 2008.

[32] WhatsApp Inc. WhatsApp. `http://www.whatsapp.com/`, 2009.

[33] Jun Zhu, Jianhua Gong, Weiguo Liu, Tao Song, and Jianqin Zhang. A collaborative virtual geographic environment based on P2P and Grid technologies. *Information Sciences*, 177(21):4621 – 4633, 2007.