Secure Communication Setup for a P2P based JXTA-Overlay Platform

Abstract— **At present time, the maturity of the P2P research field has pushed through new problems such as those related with security. For that reason, it is important to provide security mechanisms to P2P systems, since security starts to become one of the key issues when evaluating them. The JXTA-Overlay project is an effort to use JXTA technology to cater a generic set of functionalities that can be used by developers to deploy P2P applications. However, since its design focused on issues such as scalability or overall performance, it did not take security into account. This work proposes a security framework specifically suited to JXTA-Overlay's idiosyncrasies. The main features of the proposal presented in this paper include a completely modular approach which may cater to a broad set of scenarios, an effective secure key distribution method and a hybrid key authenticity scheme which balances the need for meaningful information at end-user level and simplicity at the lower middleware layers.**

Index Terms— **Peer-to-Peer, security, XMLdsig, JXTA, JXTA-Overlay, distributed systems, encryption.**

I. Introduction

Peer-to-peer (P2P) systems have become highly popular in recent times due to its great potential to scale and the lack of a central point of failure, turning to one of the main spearheads in digital ecosystems [1], [9]. Just as the popularity of P2P systems has risen, so has concerns regarding their security, specially since their decentralized nature prevents the use of a single server which may act as a trusted party for the whole network, capitalizing all security operations. As P2P applications move from simple data sharing to a broader spectrum, they become more sensitive to security threats and it becomes very important to include security mechanisms that can fit into a broad set of scenarios. Even at the cost of some impact on performance, a minimum baseline must be kept in any P2P system in order to ensure some degree of security even when system components will not act properly.

JXTA [20] (or "juxtapose") is a set of open protocols that enable the creation and deployment of P2P networks, providing applications the capability to discover and observe peers, exchange messages and publish resources. Such protocols are generic enough so they are not bound to a narrow application scope, being adaptable to a large set of application types. For that reason, they keep implementation independence and can be deployed under any programming language or lower level transport protocols.

JXTA-Overlay [8] is a JXTA-based framework. Its main goal is to improve the original JXTA protocols, increasing the reliability of JXTA-based P2P applications and enhancing its group management and file sharing capabilities. However, the design focus of JXTA-Overlay was completely concerned with system performance and did not take into account security at all, a situation which may become a great constraint under today's required standards for distributed systems.

The contribution of this paper is a modular security framework specifically suited to the characteristics of JXTA-Overlay. The proposed framework enables the setup of a secure communication environment. Our proposal fully realizes the messaging capabilities and functions of both JXTA and JXTA-Overlay and uses them in order to provide a security baseline in an invisible manner. As a result, minimum effort is necessary by application developers and end-users to deploy a security baseline. Furthermore, because of the framework's modular approach, it may be easily ported to different scenarios, according to the final application's security needs.

This paper is organized as follows. Section II provides a general overview of JXTA and JXTA-Overlay's architecture and functions, as well as some insights on the current state of its security. Section III presents the current related work on securing JXTA-based systems, so it is possible to study which approaches may be ported to JXTA-Overlay. The proposal of a basic security framework is presented in Section IV, providing a description of the global security layer's architecture and how the cryptographic data setup is performed. Section V formalizes the new set of secure primitives and functions that will enable a secure framework. In Section VI, a brief study of the impact that the security layer has on performance is presented. The short Section VII briefly provides some insights regarding how the proposed approach may be used in industrial processes. Concluding the paper, Section VIII summarizes the paper contributions and further work.

II. JXTA-Overlay Overview

JXTA-Overlay is a middleware built on top of the JXTA specification [21], which defines a set of protocols that standardize how different devices may communicate and collaborate among them. JXTA-Overlay extends the JXTA protocols with the aim to overcome some of its limitations: the need for the developer to directly manage the presence mechanism, peer group publication and message exchange. To achieve this end, JXTA-Overlay provides a set of basic functionalities, named *primitives* and *functions*, intended to be as complete as possible to satisfy the needs of most P2P applications.

A. The JXTA-Overlay network

A JXTA-Overlay network is structured in the following types of interacting entities:

End-users may connect to the JXTA-Overlay network by previously authenticating via a username and password. Once the authentication process is successfully completed, they are organized into distinct overlapping groups. It is also important to take in to account that JXTA-Overlay end-users are mobile, they may connect at different times using different client peers, as well as may be connected through several client peers at the same time. Only end-users from the same group may interact.

A *client peer* represents an application through which end-users communicate and share resources. They authenticate to a broker and forward end-user data to other client peers.

Brokers control access to the network, by requesting end-user authentication, and help client peers interact by propagating their related information. Brokers are very important, since they exchange information about all client peers, maintaining a global index of available resources, thus allowing all peers to find network services. Brokers also act as beacons for client peers which have recently gone online use to join the network. For that reason, they usually have well-known identifiers, such as a DNS name or a static IP address. Each group is assumed to have at least one broker available, even though the same broker may be shared across several groups.

All the information related to end-user configuration (username, password and group membership) is stored in a special single entity within the JXTA-Overlay network: the central *database*. Only brokers may access the database data, in order to check end-user authentication attempts and properly organize them into groups. It is assumed that some *administrator* takes care of properly configuring the database, registering new end-users. Nevertheless, JXTA-Overlay does not impose any constraint on the database architecture.

B. General architecture

The architecture of the JXTA-Overlay middleware defines three modules, which enable interaction between the different entities described in Section II-A: the Client Module, the Broker Module and the Control Module. Altogether, they form an abstraction layer on top of JXTA, as shown in Figure 1.

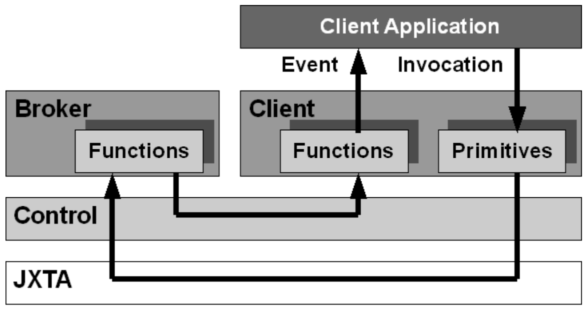

Fig. 1. JXTA-Overlay architecture:
Client Application: JXTA-Overlay application.
Client Module: Client application services.
Broker Module: Broker services.
Control Module: Intermediate JXTA-Overlay specific messaging layer.
JXTA: JXTA protocols for data exchange.

- The *Client Module* defines all necessary primitives for peer clients to join a JXTA-Overlay network and interact with other client peers and brokers. In fact, applications developed on top of JXTA-Overlay are always based on the invocation of Client Module primitives and the processing of events thrown upon primitive execution completion or message reception. Primitives comprise the following subsets: (a) peer discovery; (b) peer resources discovery; (c) resource allocation, (d) task submission and execution; (e) file/data sharing, discovery and transmission; (f) messenger; (g) peer group functionalities and (h) monitoring of peers, groups, and tasks.
- The *Broker Module* defines all the functions that client peers may call upon a broker in order to be granted access to the the network, create and publish groups or retrieve other client peers' information. Broker functions are always executed as a result of Client Module primitives and always produce a reply message from the broker to the calling client peer.
- The *Control Module* acts as an intermediate layer between the Broker and Client Modules, providing generic functionalities on regards to group management and messaging.

The Control Module provides messaging between JXTA-Overlay entities using JXTA *pipes*, a virtual communication channel between peers. Each client peer has an input pipe for each group it belongs to, so other group members may send messages using the input pipe associated to that group. Brokers have a single input pipe which is shared for all incoming messaging.

Resource information is propagated across group members by Brokers, crossing boundaries such as client peers located beyond broadcast range. Such information is formatted as JXTA *advertisements*, special metadata XML documents exchanged using the JXTA core protocols. JXTA-Overlay has a big reliance on both JXTA-defined and custom-defined advertisements. As a result, their data is critical for the correct operation of the JXTA-Overlay network. Each client peer periodically broadcasts a set of advertisements containing information regarding presence, open input pipes, capabilities, statistics and shared files. Client peers store received advertisements into a local cache, so they are able to easily retrieve the published information at a later time.

C. JXTA-Overlay and security

As previously mentioned, JXTA-Overlay's design is not concerned with security, end-user authentication via a username and password being the only exception. As a result, it is vulnerable to different security threats which may jeopardize the network. A security study must take into consideration the fact that not only entities external to the JXTA-Overlay network may try to subvert it, but also malicious end-users which have properly authenticated to a broker.

Some of the greatest security concerns in the current version of JXTA-Overlay are the following ones:

- **No privacy:** Transmitted data may be easily eavesdropped, since no encryption is provided at any layer. Even though it may be argued that data privacy in message exchanges between end-users is just an optional feature, there are some cases where privacy should not be optional, namely the initial authentication via username and password. Currently, both fields are sent as plain text, and can be easily intercepted.

- **Advertisement spoofing:** Any legitimate user may forge advertisements with no fear of reprisal. No integrity or source authenticity is enforced. False fields, such as the source client peer identifier or any other relevant information, may be added to the advertisement, which will be automatically distributed by the broker and accepted by all group members.
- **Broker impersonation:** Client peers connect to a self-proclaimed broker, but never check if it is a legitimate one. Even in the case that client peers are connecting to the proper broker address, there is no guarantee that the broker is a legitimate one, since it may be the case that traffic is being redirected to a fake broker via methods such as DNS spoofing [6].

As can be seen, some of the current JXTA-Overlay vulnerabilities are quite glaring, such as transmitting sensitive data with no real privacy. Therefore, it can be concluded that JXTA-Overlay does not provide a security baseline and needs serious improvement on this regard.

III. Related Work on Securing JXTA-based Frameworks

Before a security framework for JXTA-Overlay may be proposed, it is useful to review which are the current security mechanisms available to JXTA-based applications. From this review, it is possible to study which may prove useful and suitable to JXTA-Overlay's architecture and network setup specifics. Therefore, in this section, we provide a general security overview for JXTA applications. However, a much more complete survey may be found in [12].

As far as network access control is concerned, one of the original creators of JXTA proposes a specific trust model in [3]. Without actually recognizing a specific Certification Authority (CA) for each peer group, he proposes that rendezvous peers become the system's trust anchors, providing credentials to peers, that can be used to prove identity membership. To acquire a credential, the peer must be authorized via an LDAP (Lightweight Directory Access Protocol) [19] directory with a recognized protected password. This proposal is later extended in [16], basing the security model on a centralized Public Key Infrastructure (PKI) and a basic challenge-response protocol [26] as a means for authentication when a peer joins a peer group. Its main contribution is providing a method which peers may use in order to also authenticate the group itself.

More elaborated proposals are presented in [17], [18], based on joint authorization by multiple peers under voting schemes in order to maximize decentralization. Under these approaches, credentials are also signed certificates issued by a CA. However, access is based on an agreement protocol between several group members. The main difference between both proposals is that the latter also includes a rank system, where peers who join the group, named *"newbies"*, have the least privileges, but may rise to higher positions as they contribute to the group.

On regards to message security, the JXTA reference implementation [14] provides two mechanisms: its own definition of standard TLS (Transport Layer Security) [24] and a JXTA-specific protocol named CBJX (Crypto-Based JXTA Transfer) [5]. The former provides private, mutually authenticated, reliable streaming communications, whereas the latter provides lightweight secure message source verification, but no privacy at all.

In order to use both mechanisms, TLS and CBJX, a specific group membership service is required: the Personal Security Environment (PSE). The membership service is one of the JXTA core services, taking care of peer group access control and identity management by providing each group member with a credential. Peers may include credentials in messages exchanged between group members in order to prove identity ownership, group membership and provide a means for implementing access control in services. However, the only credentials supported by PSE are X.509 certificates [4], solely relying on Java keystores (JKS) [22] as a cryptographic module.

The current JXTA reference may also provide some degree of security in resource publication and discovery by optionally signing advertisements. No distinction between different types of advertisements is made, all become a new type of advertisement when signed: the Signed Advertisement. A Signed Advertisement encapsulates the original XML data as plain text encoded via the Base64 algorithm [7]. Signed advertisements are also constrained to the PSE membership service.

An alternative method for advertisement security is proposed in [13]. This method is based on XMLdsig [27] and can be applied to those advertisement types defined by JXTA as well as those custom made by JXTA-based applications. The resulting secure advertisement maintains its original type, instead of becoming a completely different new type of advertisement, maintaining peer interoperability.

IV. A Security Framework for JXTA-Overlay

In this section, we present a secure framework for JXTA-Overlay which provides a baseline for protecting end-user applications against the current vulnerabilities exposed in Section II-C. This framework establishes a secure communication setup which may be used as a starting point to add secure capabilities to all JXTA-Overlay primitives and functions.

In our proposal, we combine several methods of those previously described in Section III, adapting them to JXTA-Overlay's specific architecture and network setup. Client peers are protected against impersonation by using broker-issued credentials in a similar way to the approach in [3]. However, we further extend this approach to brokers, so legitimate ones may be told apart from malicious ones. We also provide an alternate lightweight method for message source authenticity between client peer endpoints. Advertisement integrity and authenticity, as well as a transparent method for key transport, is also provided by adapting the method defined in [13] to the particularities of JXTA-Overlay.

The following notation will be used from now on to describe the secure framework:

- SK_i: Peer i's secret key.
- PK_i: Peer i's public key.
- $Cred_i^j$: Entity i's credential, issued by j. It is assumed that the credential contains a public key owned by i.
- $E_{PK_i}(x)$: A string x encrypted using the PK_i by means of a wrapped key encryption scheme (such as the one defined in [2]).
- $S_{SK_i}(x)$: A string x signed using the SK_i.
- $i \longrightarrow j : \{m_1, \cdots, m_n\}$: A message sent from peer i to peer j, with the content $m_1, \cdots, m_n$.

A. Secure architecture overview

An overview of the main components in our security framework proposal for JXTA-Overlay is presented in Figure 2. All modules are located at different layers within JXTA-Overlay and JXTA's own architecture.

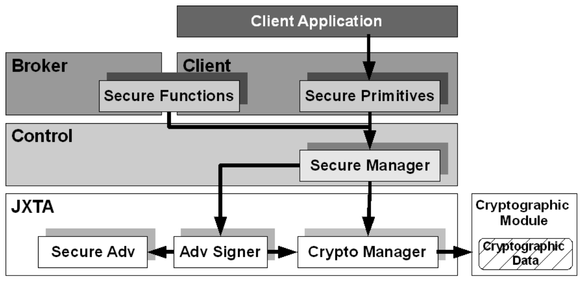

Fig. 2. Security architecture for JXTA-Overlay:
Secure Primitives and Functions: API to secure services.
Secure Manager: Main security service provider.
Crypto Manager: Key management and storage.
Adv Signer and Secure Adv Modules: Advertisement security layer.

The following modules are not JXTA-Overlay specific, but have been designed as an extension to the JXTA core protocols. As such, they may be used in any JXTA-based application:

- The *Crypto Manager* provides an abstraction layer for any cryptographic module and key management method. It enables the integration of security services with any kind of cryptographic module, such as hardware cryptographic tokens. This approach takes into account the fact that secure modules have different access methods and, in some processes require secure hardware tokens, such as smart cards [25]. The Crypto Manager is completely modular and accepts different implementations, according to the needs of the specific cryptographic module being used. Developers need only provide a proper implementation of the Crypto Manager that will support the chosen cryptographic module and credential type. This is in contrast with PSE, as exposed in Section III, which is restricted to a single type of cryptographic module, Java keystores, which can only use text password based access control.
- The *Secure Adv Module* defines the secure advertisement format, providing a method for key transport, and additional fields related to its signature. This module is based on the work proposed in the JXTA-XMLsec project [10].
- The *Adv Signer Module* manages JXTA advertisement signature and validation by interacting with the Crypto Manager. Secure advertisement management is provided such a way that there is no need to modify the standard JXTA libraries.

The JXTA-Overlay specific modules follow:

- The *Secure Manager* is the common interface to all additional security capabilities within the JXTA-Overlay Control Module, operating as a single entry point for all secure services.
- The *Secure Functions* and *Primitives Modules* just extend the base Broker and Client Modules, discussed in subsection II-B, providing a set of additional primitives and functions which take into account security considerations.

As far as the secure framework management is concerned, the end-user application developer just has to choose which primitives to use in order to create a P2P application, just as in standard JXTA-Overlay. The secure versions of JXTA-Overlay's primitives do not replace JXTA-Overlay's original insecure versions, but just complement them, leaving the final choice on which primitives to use up to the developer.

B. Communications system initial setup

In order to deploy a secure framework, JXTA-Overlay entities must be provided with cryptographic data beforehand. In our framework, such data exchange is performed using a setup divided in three distinct stages: deployment, boot and login.

1) Deployment stage: Deployment stage data is generated only once in the full system's lifecycle. Whenever a new JXTA-Overlay network is deployed, the administrator generates a public key PK_{Adm} and secret key SK_{Adm}. From both keys, also generates a self-signed credential, $Cred_{Adm}^{Adm}$, thus acting as trusted party for all peers. This is a sensible stance, since, nevertheless, the system administrator has absolute control on the legitimate end-user database.

Each broker, Br_i, is provided a well-known identifier ID_{BR_i} by the administrator (in fact, JXTA-Overlay already does this: a DNS name or static IP address), which will be the one client peers will use to connect to it. Br_i also generates a public and secret key, PK_{Br_i} and SK_{Br_i}. From PK_{Br_i}, the administrator will provide Br_i with a credential $Cred_{Br_i}^{Adm}$, by signing PK_{Br_i} and ID_{BR_i} with SK_{Adm}. Therefore, only legitimate brokers will hold a proper credential and be able to prove its ownership.

Each client peer, Cl_i, who wants to connect to the JXTA-Overlay network is provided with a copy of $Cred_{Adm}^{Adm}$ by the administrator.

It must be pointed out that all cryptographic data at this stage is created outside of the scope of a running P2P application and distributed out-of-band, since the administrator himself is a JXTA-Overlay entity but not an application. For for example, $Cred_{Adm}^{Adm}$ may be provided to Cl_i's end-user when the administrator delivers him the username and password. All broker cryptographic data is deployed along broker installation into the network.

2) Boot stage cryptographic data: Only client peers generate additional cryptographic data at boot time, immediately before going online: a key pair PK_{Cl_i} and SK_{Cl_i}. The main reason for such keys not enduring the whole node's life, in contrast with brokers, is the fact that end-users are mobile, and therefore, different end-users may use the same exact node at different stages during the client peer's lifecycle. In that case, that would mean end-users would also use the same key pair. In addition, requiring the end-user to transport and manage the key pair between nodes quickly becomes a hassle.

Once the key pair has been generated, the client peer identifier is set as a Crypto Based IDentifier (CBID), further described in section IV-C.1. At this point, it is sufficient to say that it is a method to univocally bind the identifier to PK_{Cl_i}.

3) Login stage cryptographic data: Each time an end-user logs into the network through a client peer, the broker issues a temporary credential, $Cred_{User}^{Br_i}$, containing the client peer's PK_{Cl_i} and the end-user's username. An end-user connecting to the network via several client peers will be provided several credentials, each one assigned and managed by each specific client peer. Credentials are only valid for a single end-user's session. This accounts for the fact that, since end-users are mobile, PK_{Cl_i} may be a different one each time the end-user logs into the network.

A summary of the final cryptographic data distribution after setup is shown in Figure 3. The figure legend states at which stage each particular data is initialized.

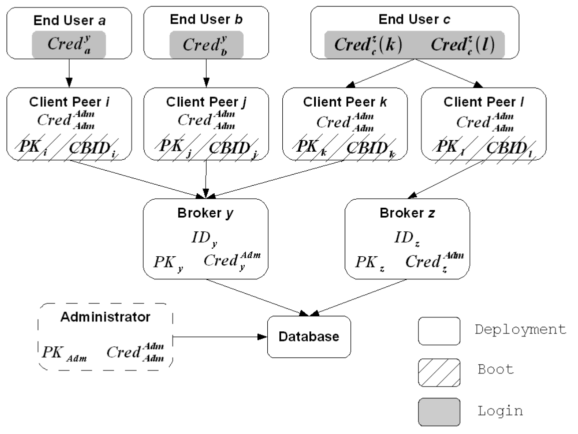

Fig. 3. Cryptographic data setup summary, by stage:
$Cred_i^j(k)$: End-user's i's credential, issued by j for peer k.
PK_i: Peer i's public key.
$CBIB_i$: Peer i's Crypto Based Identifier.
ID_i: Peer i's identifier.

C. Key distribution and advertisement security

The final step for a correct cryptographic data setup requires that all participating entities exchange their public keys, so secure protocols may be initiated at any time. However, any public key distribution method must take into account the fact that in a P2P network nodes may go online and offline at any moment. To achieve this goal, we apply the scheme defined in [13], based on XMLdsig, where the public key is included into JXTA advertisements.

Key distribution is achieved via JXTA-Overlay's standard information propagation mechanism based on advertisements, as exposed in section II-B. This approach is unobtrusive and seamlessly integrates with JXTA-Overlay's messenger primitives by making use of exactly the same core mechanisms, instead of relying on additional protocols for key distribution. Pipe Advertisements have been chosen as the ones which hold peer credentials. The reasons for this choice are twofold.

First of all, JXTA-Overlay's messaging capabilities between group members completely rely on input pipes, as explained in section II-B. Client peers cannot exchange messages unless they have each other's Pipe Advertisement. Consequently, by publishing keys using this advertisement type, it is also always guaranteed that both parties have each other's public key before any message exchange may actually begin.

Additionally, the chosen scheme allows the signature of Pipe Advertisements, providing effective protection against advertisement forgery (during transport or even when already stored at the local cache), a vulnerability exposed in section II-C. It must be heavily remarked the importance of each client peer's JXTA input pipe in JXTA-Overlay. Once a broker has granted access to a client peer, absolutely all incoming messages are received via this pipe. Therefore, it is very important to secure the distribution of each client peer's Pipe Advertisement to avoid that a rogue peer may publish a forged advertisement, claiming that its own input pipe is assigned to some other Peer ID. In such scenario, all incoming messages towards that Peer ID would be automatically redirected to the rogue peer. By signing Pipe Advertisements, advertisement fields cannot be tampered and it is easy to detect spoofers.

However, any secure scheme on JXTA-Overlay advertisements must take into account that their fields are used to specify which client peer and end-user is the input pipe owner. Therefore, it is important to keep such field visible, so they may be easily located by the indexing services. The chosen approach accomplishes this requirement. Another crucial advantage, in contrast with JXTA's Signed Advertisement, is its capability to become invisible to standard JXTA-Overlay operation, instead of adding a new advertisement type, completely opaque to advertisement indexing and retrieval services.

A sample signed Pipe Advertisement is shown in Listing 1 (some fields have been shortened to improve readability).

1) Guaranteeing key authenticity: An additional requirement for key distribution is that it must always guarantee public key authenticity, so anyone may check whether an endpoint is the legitimate owner of any particular public key. In JXTA-Overlay, messaging may occur between two different endpoint types: client peers and end-users. The former comprises control messaging such as advertisement propagation whereas the latter is concerned with direct communication exchanges such as chatting or file sharing. Hence, key ownership must consider

XML Listing 1 - Signed Pipe Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PipeAdvertisement xmlns:jxta="..." ID="signed">
  <Id>urn:jxta:uuid-59...904</Id>
  <Type>JxtaUnicast</Type>
  <Name>urn:jxta:uuid-59...B03</Name>
  <Desc>username</Desc>
  <Signature xmlns="...">
    <SignedInfo>
      <CanonicalizationMethod
       Algorithm="..."/>
      <SignatureMethod Algorithm="..."/>
      <Reference URI="#signed">
        <Transforms>
          <Transform Algorithm="..."/>
        </Transforms>
        <DigestMethod Algorithm="..."/>
        <DigestValue>/dL...yo=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
      dg...NGg=
    </SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509SubjectName>
          CN=username,O=Overlay
        </X509SubjectName>
        <X509Certificate>
          MII...Qw==
        </X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</jxta:PipeAdvertisement>
```

both types of endpoint data exchanges. It must also take into account the fact that a single end-user may be connected to the JXTA-Overlay network using several client peers at the same time.

We accommodate to these constraints by using a joint scheme were different authenticity mechanisms are used depending on the endpoint type. On one hand, since exchanges between client peer endpoints are very frequent, a lightweight method is desirable. Thus, CBIDs are used. On the other hand, exchanges between end-users require some method that provides additional information that may be presented to the user, such as the source username and an easily recognizable identifier of a trusted entity, so he may decide whether to accept the exchange or not. Therefore, signed credentials are used in this case. The schematics of this key authenticity method are shown in Figure 4.

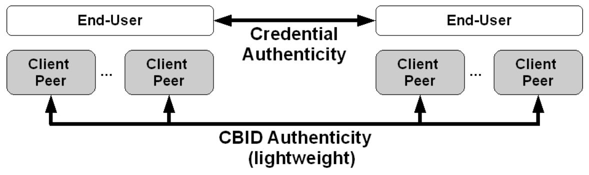

Fig. 4. Key authenticity model:
End-user: JXTA-Overlay application user.
Client peer: JXTA-Overlay application.

The concept of CBIDs, or Statistically Unique and Cryptographically Verifiable IDs (SUCV IDs), was initially conceived for IPv6 addressing in order to solve the issue of address ownership [23]. We adapt this scheme to our security framework by applying it to the client peer identifier. The client peer public key PK_{Cl_i} is bound to the client peer identifier by applying a pseudo-random function on the public key. The result is henceforth used as the JXTA peer identifier. Secure messages between client peer endpoints are signed using SK_{Cl_i}. In order to check CBID ownership, the message's signature is validated. If validation is correct, it is proved that the source peer holds the associated private key. Then, the validating public key is used to generate the source peer identifier, the CBID. If the obtained identifier is the same as the claimed one, the message is deemed authentic.

Key authenticity via credentials is checked by verifying the credential's signature against the issuer's public key, validating the full certificate path: end-user, broker and administrator (being the latter trusted by all entities in the system). The brokers' credentials are available to client peers, since they are also distributed within their Pipe Advertisements. The administrator's single credential is provided at deployment, as explained in section IV-B.

It must be pointed out that this hybrid approach is necessary, since CBIDs cannot be used for end-user endpoint data exchanges nevertheless. It is not possible generate one that conforms to a particular username. In addition, CBIDs which could be considered secure enough, being about 40 hexadecimal digits long, would not be easy to remember for end-users.

V. Secure primitives and functions

The secure framework is based on a set of secure primitives, handled by the Secure Manager module, which coexists with the original JXTA-Overlay modules. Each original primitive may be mirrored in a secure version, however, because of the sheer number of different primitives, about 122 primitives and 84 events related to different functions, we will only focus on two important primitive types: *discovery primitives* and *messenger primitives*.

A. *Secure discovery primitives*

Discovery primitives negotiate how client peers locate and connect to a broker when they want to join the JXTA-Overlay network and retrieve miscellaneous client peer information. The proposed secure extension exclusively focuses on the initial interactions with the broker in order to join the JXTA-Overlay network, and the eventual cryptographic data setup and exchange.

Joining the network via the broker is divided into two distinct steps, each one handled by a specific primitive:

- *connect*: Locates a broker and waits for a connection to open.
- *login*: Authenticates the current client peer end-user by sending a username and password that will be checked against the database.

The secure extension maintains the separation in two steps: the *secureConnection* and *secureLogin* primitives.

1) secureConnection: The description of the connection process, initiated by the primitive invocation, follows:

1) The client Cl waits for a broker Br to become available and initiates the connection attempt.
2) Cl chooses a byte array, $chall$, as a random challenge.
3) $Cl \longrightarrow Br$: $\{chall\}$
4) Br generates a sufficiently long random session identifier sid, and stores it.
5) $Cl \longleftarrow Br$: $\{sid, S_{SK_{Br}}(chall), Cred_{Br}^{Adm}\}$
6) Cl checks the authenticity of $Cred_{Br}^{Adm}$ by verifying its signature using PK_{Adm} (contained in the administrator's credential, $Cred_{Adm}^{Adm}$).
   - If its is not authentic, it can be concluded that Br is not a legitimate Broker.
7) Cl checks $S_{SK_{Br}}(chall)$ using PK_{Br} (which is contained within $Cred_{Br}^{Adm}$).
   - If signature validation fails, it can be concluded that Br does not possess SK_{Br}, and thus is an impersonator.
8) If all checks succeed, it can be concluded that Br is a legitimate Broker.
9) Cl stores sid and $Cred_{Br}^{Adm}$.

The main attack at the Broker location and connection stage is Broker impersonation. The digital signature mechanism in step 5 provides Broker authenticity, since only a legitimate Broker has a public key PK_{Br} in a proper credential, $Cred_{Br}^{Adm}$, issued by the Administrator. Additionally, the challenge-response approach in steps 2-5 avoids replay attacks, where an attacker patiently intercepts and stores signatures generated by the Broker at step 5 and tries to reuse them in future exchanges, passing as the Broker. Each time a *secureConnection* primitive is called, different data must be signed by the Broker.

2) secureLogin: Once the Broker Br's credential has been retrieved, and its authenticity has been established, it is possible to use the *secureLogin* primitive to actually join the JXTA-Overlay network. As required by JXTA-Overlay's entities and architecture, a username and password are provided by the client application's end-user.

The description of the underlying protocol follows:

1) Cl generates the login request $req = S_{SK_{Cl}}(username, password, PK_{Cl})$
2) Cl retrieves PK_{Br} and sid, obtained during the *secureConnection* primitive call.
3) $Cl \longrightarrow Br$: $\{E_{PK_{Br}}(req, sid)\}$
4) Br decrypts the message using SK_{Br}.
5) Br checks if sid is currently stored. If that is not the case, login is aborted. Otherwise, Br no longer stores sid and the login process continues.
6) Br checks username and password matching according to JXTA-Overlay's standard procedures (for example, a secure back-end database connection).
   - If they do not match, it can be assumed that Cl's end-user is an impersonator and login is aborted.
7) Br checks key authenticity against the claimed client peer identifier according to the mechanism described in Section IV-C.1.
   - If the check fails, it can be concluded that the request was not received from a client peer with the claimed identifier. The login attempt is aborted by Br.
8) If both checks were correct, Br generates a credential $cr = Cred_{Cl}^{Br}$, containing PK_{Cl} and Cl's current end-user's username.
9) $Cl \longleftarrow Br$: $\{cr\}$
10) Cl stores the received credential into its cryptographic module. From now on, Cl's end-user may use cr as proof of identity, until cr's expiration.

At this stage, the main security concerns are attackers trying to eavesdrop the end-user's username and password, and end-user or client peer impersonation. As far as the former is concerned, the new secure primitive encrypts data, in contrast with the original one, which did not, and thus data cannot be easily intercepted. The cryptographic key used to encrypt the data was retrieved during the *secureConnection* primitive call, and thus can be considered authentic (from a legitimate Broker).

Since end-user authenticity is still provided using a username and password pair, it might be possible for an attacker to reuse past messages to try to impersonate an end-user even when the message content in step 3 cannot be decrypted. It would be enough that it contained a valid username and password that will be accepted by the broker. However, the changing session identifier, sid, avoids this in step 5.

Client peer impersonation is subverted by the CBID mechanism enforced in step 7. Since each client peer identifier is univocally linked to is public key, an attacker cannot try to publish a Pipe Advertisement which contains any chosen identifier, but the attacker's public key. Such maneuver is easily detected, guaranteeing client peer key authenticity.

B. Secure messenger primitives

Messenger primitives define how to directly exchange simple text messages between end-users. Once the cryptographic setup has established key and credential information for all participating entities, it is finally possible to secure data exchanges by providing a secure version of each primitive. The two main messenger primitives are:

- *sendMsgPeer*: Sends a simple message to some other client peer.
- *sendMsgPeerGroup*: Sends a simple message to all members of a group. It is actually resolved by iteratively calling *sendMsgPeer*.

The secure versions of both primitives provide lightweight privacy, data integrity and message source authentication in a stateless, best effort method. This is in contrast, for example, with JXTA's secure pipes, which rely on TLS and require some previous negotiation between endpoints each time a message exchange is initiated, as explained in section III.

1) secureMsgPeer and secureMsgPeerGroup: The necessary steps for some end-user connected to client peer Cl_i to send a simple text message to another one connected to Cl_j are:

1) Cl_i retrieves Cl_j's Pipe Advertisement. This step also happens in the original, insecure, primitive and thus means no additional burden.
2) Cl_i validates the advertisement signature in order to ensure that it has not been compromised, using the method described in [13].
   - If the signature does not validate, the advertisement has been tampered, and is deemed invalid. If the message is sent nevertheless, no guarantees can be made on regards to its security.
3) Cl_i retrieves PK_{Cl_j} from the signed advertisement's enclosed credential, $Cred^{Br}_{Cl_j}$.
4) $Cl_i \longrightarrow Cl_j$: $\{E_{PK_{Cl_j}}(m, S_{SK_{Cl_i}}(m))\}$
5) Cl_j decrypts the message using SK_{Cl_j}.
6) Cl_j repeats steps 1, 2 and 3.
7) Cl_j validates the message signature using PK_{Cl_i}, obtained via Cl_i's signed advertisement.

The *secureMsgPeerGroup* primitive just iteratively uses the *secureMsgPeer* to send the same message to a group of peers.

As a standard message exchange between end-user endpoints, these primitives are mainly subject to data eavesdropping and end-user impersonation. The former is avoided by encrypting data using a wrapped key approach, whereas the latter is provided by digitally signing the data (which, additionally, ensures data integrity). In both cases, the opposite endpoint's public key is necessary in steps 2 (signature validation) and 4 (encryption and signature generation). However, since both public keys are contained in the credentials provided to the endpoints by the Broker in step 8 of the *secureLogin* primitive call, they can be considered legitimate, and not from an impersonator.

VI. Security Computational Cost Analysis

Security always comes at a cost in protocol efficiency by adding some degree of overhead. To assess the impact on performance, two different sets of scenarios have been taken into account in order to run a set of tests: overhead in the expected time until a client peer joins the JXTA-Overlay network and delay in simple message transmission. Additionally, advertisement publication overhead because of key distribution has also been calculated.

All tests have been run using a PC with a 1.20 GHz Intel Pentium M processor and 1 Gb of RAM under Ubuntu 8.10 and SUN's Java Runtime Environment version 1.6.0.10 (which includes the Java Cryptographic Extension, JCE) . We decided to use a computer which is below today's average standards to assess the impact of using JXTA's security mechanisms on low end machines. Furthermore, both the client peer and the broker where at the same 100 Mbps LAN segment, making network latency almost insignificant. Under this configuration, we assume an even worse scenario for overhead calculation, since network latency has almost no effect on the final results.

The analysis of the resulting average overhead when joining the network via the *secureConnection* and *secureLogin* primitives is shown in Table I. Each row represents a particluar set of the steps specificed in sections V-A.1 and V-A.2 for both primitives, respectively. Each row also points out if the steps are executed at the client peer side (*Cl*) or at the broker side (*Br*).

In this table, we calculate overhead as:

$$Overhead = 100 * \frac{cryptographyTime}{totalTime}\%$$

Primitive	Side	Steps	Overhead
secureConnection	*Cl*	Steps 1-3	1.83%
	Br	Steps 4-5	1.72%
	Cl	Steps 6-9	11.80%
secureLogin	*Cl*	Steps 1-3	8.96%
	Br	Steps 4-9	41.04%
	Cl	Step 10	16.41%
Total Overhead			**81.76%**

TABLE I
Cryptographic setup overhead

As can be seen from the results, the final overhead may be considered quite high from an absolute standpoint, since the required time for connecting to the network is almost doubled. In fact, since no broker authentication existed in the original version of JXTA-Overlay, the *secureConnection* primitive can be considered as full overhead. However, we must take into account that these results are in comparison to a scenario where no security exists at all. Furthermore, it must also be taken into account that client peer login just happens once for the full session. Therefore, the final overhead actually has a very low repercussion on overall system performance.

As far as overhead in secure messaging is concerned, using the *secureMsgPeer* primitive, it has also been tested for different data lengths, as shown in Figure 5. Even though the overhead is a bit high for small messages, the reasons being the same as the case of the network join process, it quickly falls as network latency becomes more relevant.

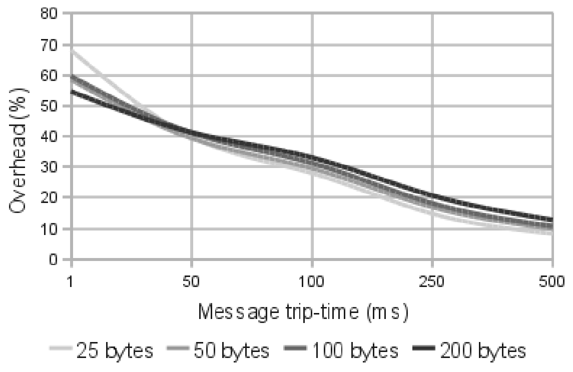

Fig. 5. *secureMsgPeer* primitive overhead.

Finally, the cost on performance because of the new Pipe Advertisement format, allowing key distribution, has been calculated. This cost is twofold, in advertisement data size and in processing time, because of XML signature generation and validation.

In order to calculate the data size overhead in advertisement publication, it must be taken into account that JXTA-Overlay client peers make heavy use of them in order to publish all available resources. Different types of advertisements are periodically published at different default time intervals in seconds, depending on the resource type. Therefore, it is possible to calculate the advertisement publication data rate for a client peer. This data, for each advertisement type, is shown in Table II, based on a a client peer with no available resources, which would be the lowest possible data rate scenario.

Advertisement type	Interval	Size	Rate
Peer	10	1081	108.1
Peer Group	10	310	31
Pipe	5	409	81.8
Info	5	953	190.6
Files	25	672	26.88
Criterium	15	5759	383.93
Stats	30	18312	610.5
Total Data Rate			1432.71

TABLE II
ADVERTISEMENT PUBLICATION DATA RATE. (INTERVAL IN SECONDS, SIZE IN BYTES, RATE IN BYTES/SEC)

The use of signed Pipe Advertisements increases the data rate from 81.8 (third row) up to 407.2. The final result is an overhead of about 22.7%, quite acceptable. It must also be taken into account that about 12% of this overhead is because of credential size. With smaller credentials, overhead may decrease. Furthermore, this is our worst case scenario. A peer sharing a lot of resources will incur in higher data rates, also decreasing overall overhead. Nevertheless, for sensitive environments, JXTA-Overlay allows overriding default values and fine tune Pipe Advertisement publication intervals.

On regards to processing time, Pipe Advertisements are only signed once, at peer boot time. Therefore the impact of signing during standard operation was found out to be negligible. At boot time, the impact is an average delay of 205 ms, which is very low (just finding a broker is about 2 orders of magnitude higher). Similarly, each advertisement needs only be validated when it actually has to be used to exchange data with another endpoint. Therefore, even though a peer may receive many advertisements, signature validation only impacts message sending primitives and functions, since the signature (and the advertisement content) is ignored until that point. The average delay because of validation is about 12 ms, which has different degrees of meaningfulness depending on the primitive used. For the standard sendMsgPeer primitive (see Section V-B), considering a 0 network latency, that amounts to a 5-10% overhead for a message 25-200 bytes long.

VII. APPLICATION TO INDUSTRIAL AND MANUFACTURING SYSTEMS AND PROCESSES

Even though our proposal uses JXTA-Overlay as the base scenario, some of the methodological approaches may be used in scenarios which go beyond pure desktop applications, such as industrial systems and processes. In some instances, they can be considered equivalent to a P2P network, where entities are organized into groups and collaborate to achieve a common end, such as in cellular manufacturing, with robots being akin to peers [11]. In these kind of scenarios, when hardware is being directly controlled, security becomes even more relevant. As far as this kind of scenario is concerned, in fact, our security proposal is currently being used as a means for secure automated interaction of SmartBox hardware devices [15].

Our generic methodology relies on a security baseline of data privacy and authentication, based on the division of basic operations in primitives and functions and then adding a new secure suit which may co-exist with, but not necessarily replace, them. This minimizes the impact on the developer. In addition, an effort is made to keep the core protocols unchanged, being actively reused at lower architecture layers, so secure and insecure entities may co-exist up to certain point. Finally, a joint credential scheme provides a trade off between useful human-readable information and efficient lower layer data. Thus, we can achieve the goal of a secure subsystem which operates in the most invisible manner, no matter which deployment scenario.

VIII. CONCLUSIONS

This paper proposes a security framework for cryptographic data setup in order to secure JXTA-Overlay communications. From this setup, it is possible to secure primitives and functions. As an example, a secure version of the discovery and messenger primitives has been presented. Apart from providing a baseline for the deployment of security mechanisms in JXTA-Overlay, which up to now had not been considered in its design, the main contributions of the chosen approach are threefold.

First of all, an effective framework for secure key distribution is provided, by securing Pipe Advertisements and using standard JXTA-Overlay procedures for key publication and update, guaranteeing that keys are always available whenever messages must be exchanged. As a result, key distribution becomes invisible to the JXTA-Overlay Control Module and client peers which use the secure framework and those who do not may coexist in the same JXTA-Overlay network. This is in contrast with JXTA's current approaches, which require that every single peer within the same group agrees to deploy security protocols.

Second, key authenticity is provided by a combination of CBIDs and signed credentials. The net result of this approach is that, on one hand, those cases where no end-user intervention is necessary can be resolved in a lightweight manner using CBIDs. On the other hand, it is also possible to provide meaningful information to the end-user in those circumstances where his intervention may prove necessary or helpful, by providing a user credential. The proposed method also minimizes the amount of effort required for end-users in order to manage cryptographic data.

Finally, the proposed framework is completely modular and can be adapted to different scenarios suitable to the application developers' needs. This is also an improvement over the security mechanisms originally provided by JXTA, which constrain end-user applications to a very specific credential and cryptographic module type.

Further work includes using the proposed security framework to define new secure primitives for those interactions which are still deemed sensitive to attacks, in a way that they complement existing ones, but not forcibly replace them. Of special note are those of the executable set of primitives, related to remote code execution.

REFERENCES

[1] A. Waluyo, W. Rahayu, D. Taniar, B. Srinivasan. "A Novel Structure and Access Mechanism for Mobile Broadcast Data in Digital Ecosystems", 2009 *to appear in IEEE Trans. on Industrial Electronics*, vol. 56.

[2] B. Kaliski and J. Staddon, "PKCS#1: RSA cryptography specifications. version 2.0", 1998.

[3] B. Yeager, "Enterprise strength security on a JXTA P2P network", 2003 *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P'03).*

[4] CCITT, "The directory authentication framework. recommendation", 1988.

[5] D. Bailly, "CBJX: Crypto-based JXTA (an internship report)", 2002.

[6] D. Sax, "DNS spoofing (malicious cache poisoning)", 2003, http://www.sans.org/rr/firewall/DNS_spoof.php.

[7] Ed. S. Josefsson, "IETF RFC 3548: the base16, base32, and base64 data encodings", 2003, http://www.ietf.org/rfc/rfc3548.txt.

[8] F. Xhafa, L. Barolli, T. Daradoumis, R. Fernndez and S. Caball, "JXTA-Overlay: An interface for efficient peer selection in P2P JXTA-based systems", 2009, *Computer Standards and Interfaces*, vol. 31, no. 5, pp. 886 – 893.

[9] G. Briscoe and P. De Wilde. "Digital Ecosystems: Evolving service-orientated architectures", 2006, *1st International Conference on Bio inspired Models of Network, Information and Computing Systems*, vol. 275, no. 17.

[10] J. Arnedo-Moreno, "Project JXTA-XMLsec", 2009, http://kison.uoc.edu/research

[11] Y. Yang, X. Dfago, M. Takizawa, "Self-stabilized Flocking of a Group of Mobile Robots under Memory Corruption, 2009, *International Conference on Network-Based Information Systems*, pp.532-538.

[12] J. Arnedo-Moreno and J. Herrera-Joancomartí, "A survey on security in JXTA applications", 2009 *Journal of Systems and Software*, Volume 82, Issue 9, pp. 1513–1525.

[13] J. Arnedo-Moreno and J. Herrera-Joancomartí, "Persistent interoperable security for JXTA", 2008, in *Proceedings of the Second International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC'08)*, pp. 354–359, IEEEPress.

[14] "JXTA 2.5 RC1", 2007, http://download.java.net/jxta/build.

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

[16] L. Kawulok, K. Zielinski, and M. Jaeschke, "Trusted group membership service for JXTA", in *Computational Science (ICCS'04)*, 2004, Lecture Notes in Computer Science Volume 3038.

[17] L. Yunhao and H. Jinpeng, "Access control in peer-to-peer collaborative systems", 2005 *First International Workshop on Mobility in Peer-to-Peer Systems (MPPS)*, pp. 835–840.

[18] M. Amoretti, M. Bisi, F. Zanichelli, and G. Conte, "Introducing secure peer groups in SP2A", 2005, *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pp. 62–69.

[19] M. Wahl, T. Howes, and S. Kille, "Lightweight directory access protocol (v3)", 1997, http://www.ietf.org/rfc/rfc2251.txt.

[20] SUN Microsystems, "Project JXTA", 2001, http://www.jxta.org.

[21] SUN Microsystems, "JXTA v2.0 protocols specification", 2007, https://jxta-spec.dev.java.net/nonav/ JXTAProtocols.html.

[22] SUN Microsystems, "Java Cryptography Architecture (JCA)", 2008, http://java.sun.com/javase/6/docs/technotes/ guides/security/crypto/CryptoSpec.html.

[23] T. Aura, "Cryptographically generated adresses (CGA)", 2005, http://www.ietf.org/rfc/rfc3972.txt.

[24] T. Dierks and C. Allen, "IETF RFC 2246: The TLS Protocol Version 1.0", 1999, http://www.ietf.org/rfc/rfc2246.txt.

[25] W. Juang, S. Chen, H. Liaw. "Robust and Efficient Password-Authenticated Key Agreement Using Smart Cards, 2008, *IEEE Trans. on Industrial Electronics*, vol. 55, no. 6, pp. 2551-2556.

[26] W. Simpson, "PPP challenge handshake authentication protocol (chap)", 1996, http://tools.ietf.org/html/rfc1994.

[27] W3C, "XML-signature syntax and processing", 2002.