Reducing Security Overhead to Enhance Service Delivery in Jini IoT

Sung-Ki Kim, Byung-Gyu Kim, and Byoung-Joon Min

1School of Mechanical and ICT Convergence Engineering, Sun Moon University, 100 Kalsan-ri, Tangjeong-myeon, Asan-si 336-708, Republic of Korea
2Department of Computer Science and Engineering, Sun Moon University, 100 Kalsan-ri, Tangjeong-myeon, Asan-si 336-708, Republic of Korea
3Department of Computer Science and Engineering, Incheon National University, 119 Academy-ro, Yeonsu-gu, Incheon 406-772, Republic of Korea

Correspondence should be addressed to Byoung-Joon Min; bjmin@incheon.ac.kr

Received 31 July 2015; Accepted 28 September 2015

Copyright © 2015 Sung-Ki Kim et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Jini security framework that is being maintained in Apache River project requires roughly 3 sequential tasks in every connection to cope with security threats in communication between clients and servers in the IoT sensor application. These tasks are sequentially proxy preparation, certification of identity and its credential for mutual authentication, and cryptographic operation for session encryption. Since the proxy preparation task is time spent on preparing a secure and trustable stub for both client/server sides, it is not one of the substantial communication-delay factors. We propose a method of improving proxy preparation reducing service connection delay by completing the preparation of mutual authentication data and session key while on proxy preparation task. Through experiments on a test-bed, we have confirmed that service connection delay time can be reduced by about 2.6 times in proportion to the message sizes through our proposed method. Our work presents an approach to reduce the additional overhead that would be paid for by applying the Jini security framework to the development of sensor application based on Jini IoT.

1. Introduction

The advances in embedded systems, wireless communication, and sensor technologies made smart sensor networks be the underlying infrastructure to support building of IoT environment. In general, smart sensors comprising IoT are low-powered devices with wireless communication, decent computing power, and various sensing capabilities [1]. However, because they have limited resources, a sensor node itself in a smart sensor network serves as a single service to just react to a variety of ambient conditions in the real environment [2]. Therefore, in order to create values from these distributed sensors, it is necessary to develop a service-oriented framework to support the implementation of ubiquitous computing paradigm [3] on distributed sensor networks (DSNs) such as [4].

The service-oriented framework aims to support the implementation of service-oriented systems that make service available and easily accessible through standardized protocols without having to worry about the underlying infrastructure, development models, or implementation details. The service-oriented framework, in general, includes a service-oriented architecture (SOA) that defines how software components are organized on hardware platform of networks and programming model describing how to write programming codes [5].

The research works in [2, 6–12] address the service-oriented framework to support building of the ubiquitous service environment on DSN. In [2], Pereira et al. analyze the problems of existing service-oriented framework to support the development of WSN application and present a P2P Jini framework for truly service-oriented WSN as a solution to overcome their problems. The research work [13] presents a service-oriented framework that relies on Jini technology in order to support the development of plug and play features so that new sensor is readily made available to the user.

Jini [14], also called Apache River [15], is a java-based middleware supporting sharing of resources such as
ubiquitous devices and software on networks while it copes with the heterogeneity of the lower levels such as the various types of devices or communication protocols. Jini provides a mechanism that discovers available services through the lookup services and make a connection to the services that clients requested.

The security in IoT sensor application is a major concern as we have to keep in mind whether data exchanged in IoT sensor applications are authentic and confidential and have not been tampered with. Therefore, it is very important to ensure the trustable discovery of distributed services, the secure connection to the services, and the security of service codes in IoS based on smart sensor networks because of having more attack vectors that come from their distributed nature, limited resources, and the reason that we cannot guarantee physical security of sensor nodes.

Jini 2.x Security Framework [16] is being maintained in Apache River project and requires roughly 3 sequential tasks in every connection to cope with security threats in communication between clients and servers. The first one is proxy preparation, and it requires trust verification, a process to verify the trust of downloaded service proxy, and also requires the composition of granting permissions for method invocation as well as security-related constraints. The second task requires the certification of identity and its credential for mutual authentication for subjects actually using a stub. The third task is cryptographic operation to ensure the integrity and confidentiality of sessions. The second and third tasks are the same as a process required by TLS/SSL communication based on the existing X.509 certificate.

Since the proxy preparation process means time to prepare a secure and trustable stub for both client/server sides, it is not one of the substantial communication-delay factors. Through experiments, we confirmed that the 2nd and 3rd tasks have connection-delay overheads consumed for the existing SSL communication for certificate of identity and its credential.

In this paper, we propose a method reducing communication delay substantially by preparing authentication information and session keys in the proxy preparation process in advance for mutual authentication. Our approach has two dimensions. One is for service provider to insert their public key certificates issued by the authentication server into the service proxy when Jini service providers export service proxy. The other is for client to provide the service provider with their public key certificates when asking the service provider for a trust verifier through the bootstrap proxy. In this manner, authentication information and session keys are prepared in the proxy preparation process. The prepared authentication information is used to certify the subject of substantial stub use between the client and the server and provide a seed of creating encryption keys for the confidentiality of sessions.

Section 2 in this paper discusses related works on the Jini system security. Section 3 analyzes the service proxy preparation process of Jini security framework. Section 4 provides our Jini service security architecture for the improvement of proxy preparation and discusses our method. Section 5 discusses experiment methods and results, and Section 6 concludes the paper.

2. Related Work

The Jini service environments overcome the heterogeneity of devices by using the JVM (Java Virtual Machine). System development in java in order to overcome the heterogeneity of devices reduces the burden of system development, but it may provide attackers with security holes that will help to exploit flaws in different types of devices comprising the system. For example, a method to hide attacker’s intents in native code in which a Java runtime code call through JNI (Java Native Interface) is an untouchable security flaw in java-based systems [17]. The reason is that no technologies based on java language can guarantee the security of native code below JVM layer but they can guarantee the security of code above it. The Jini service environment sets free the discovery, addition, and modification of services on ubiquitous networks on the basis of mobile codes such as service proxies while they bear these risks. Therefore, the Jini systems must have a mechanism not only to provide security services such as enforcement of authentication and authorization among the Jini entities and ensuring of confidentiality and integrity for exchanged messages but also to respond to intrusion.

Some typical research works on the improvement of Jini system security are as shown in [18–20]. The following shows some security requirements for Jini system commonly suggested by these works:

(i) establishment of trust between clients, service proxies, and servers,
(ii) control on service access,
(iii) protection of clients from mobile codes,
(iv) assurance for the confidentiality and integrity of communication channels.

Reference [18] suggests a Jini service architecture to solve the establishment of trust between Lookup servers, service proxies, and servers and the requirements for access control, by implementing service in charge of authentication and permission-granting in the center of Jini federation. Reference [19] suggests a Jini security architecture and a method that establishes an authentication and authorization chain that connects server, service proxy, and user by using the certificate of SPKI (Simple Public Key Infrastructure). Reference [20] suggests a Jini security architecture similar to the architecture of [18], but the security mechanism is different from those of other researches, in a way of providing confidentiality and integrity of communication channels. Besides, to provide security transparency for the implementation of a client, a great deal of security function depends on service proxy. Works in [18, 19] depend on application implementation for security provision and have no security transparency in clients.

The security problem of service proxy codes is based on the assumption of works in [18–20] that trust the security of service proxy codes authenticated.
In the provision of security in message exchange, [18] depends on SSL-based RMI communication implemented with Java open source, while [19] depends on TSL-based RMI communication implemented with Java Standard JSSE (Java Socket Security Extension) [21]. In [20], every time a service proxy gets connected to a server, shared session keys are generated with DH (Diffie-Hallman) key exchange algorithm, through which confidentiality is provided. In addition, the integrity of messages is protected with HMAC-MD5 algorithm.

3. Proxy Preparation Task in Jini Security Framework

The Jini security framework requires three sequential tasks in every connection to cope with security threats in communication between clients and servers in IoS environment. These three tasks are performed in order of (1) proxy preparation, (2) certification of identity and its credential for mutual authentication, and (3) cryptographic operation for secure session. This section outlines the proxy preparation task only because the rest of the tasks are the same as a process required by TLS/SSL communication based on the existing X.509 certificate.

3.1. Verification of the Source of Service Proxy Objects. A Jini system consists of the lookup server (lookup service provider as shown in Figure 1), clients, and servers (remote service provider as shown in Figure 1) in which service codes are implemented. The servers and clients discover the presence of lookup servers in the ubiquitous network environment by performing the discovery protocol. After discovery of any lookup server anywhere, the servers and clients download a lookup service proxy that the lookup server provides. Afterward, the servers register their own service proxies with the lookup server and the clients discover a set of available service implementations through the lookup service proxy. At this time, if a client selects a specific service of available services, the client gets a service proxy that the corresponding server had registered from the lookup server (service proxy download as shown in Figure 1). Subsequently, the clients call the remote service implementation through the downloaded service proxy.

All request/response messages are transferred to be transparent to the lower layer protocols by using Java RMI (Remote Method Invocation) protocols.

Figure 1 shows a process of establishing trust, indicating that downloaded service proxy is secure in Jini security framework. Lookup service proxy objects downloaded by a client from the Lookup service contain URL indicating the location of data instance and class files needed for service.

In general, service providers place class files or class archives (i.e., jar) on the HTTP server. Then, a client gets these service proxy objects through HTTP.

Since proxy codes may be imported from an unauthenticated source, the proxy trust verification is composed of two different aspects. One is to confirm if service proxy objects come from a legitimate source, and the other one is to confirm if the proxy objects are originally exported by service providers.

As a means to verify the former part, Jini system uses HTTPMD URL. Since hash values are included in URL marked in a Lookup service proxy object, proxy objects can be trusted when a hash value calculated for the class file received by a client is equal to the hash value attached to URL.

3.2. Trust Verification for the Implementation of Service Proxy. When the client has confirmed if the source of service proxy objects downloaded is trustable or not, the next process is to verify if code implementation of service proxy objects is secure or not.

The client obtains a service proxy object composed of local codes completely trustable, called “Bootstrap Proxy” from the downloaded service proxy object (② as shown in Figure 1). To support this process, the service proxy should be implemented by a special method, ProxyTrustIterator getProxyTrustIterator(), in addition to methods defined in its service interface.
The client discovers this method inside the proxy object by using java.lang.reflect class. When these methods are invoked by the client, ProxyTrustIterator makes a bootstrap proxy object ultimately. Then, the bootstrap proxy invokes the method: TrustVerifier.getVerifier() throws RemoteException. This method is used to execute remote call-back that requires TrustVerifier object from the server (C as shown in Figure 1). The returned TrustVerifier object is an object verifying the trust of service proxy as a local object delivered to the client. It verifies if the service proxy delivered to the parameter of the method below is trustable or not (D as shown in Figure 1):

```java
boolean isTrustedObject(Object jiniProxy, TrustVerifier.Context context);
```

Here, the parameter of TrustVerifier.Context is used to deliver necessary contexts to the TrustVerifier object. Jini security framework provides a proxy trust algorithm based on the equality of object implementation contents (semantic). In other words, TrustVerifier object offered by the server has a single canonical proxy instance embedded, and TrustVerifier judges the conformity by comparing the proxy object received by the client and this canonical proxy instance. When they are conformed, it judges that the downloaded service proxy is trustable.

### 4. Improvements of Proxy Preparation

This section presents our approach to reduce the unnecessary time cost that would be paid for by applying the Jini security framework to the development of sensor application based on Jini IoT.

Since the Jini security framework is just a framework to support the development of secure application based on Jini technology, it does not present any tailored architecture for specific security purposes. Therefore, it is needed to design an architecture that is acceptable in the real application field and helps our approach. In this section, our architecture for supporting improving of the proxy preparation and an improved proxy preparation are presented.

#### 4.1. Proposed Architecture for Improving Proxy Preparation

To improve the proxy preparation process and to support secure communication, we propose a security architecture for improving proxy preparation as in Figure 2.

In order to ensure the trustable discovery and publishing of distributed services based on smart sensor networks, when clients and service providers obtain a lookup service proxy by accessing lookup server (LUS), they need to get an access key to the service. For this purpose, we designed a Jini service that authenticates and authorizes accesses to all services including the lookup service in a Jini system based on DSNs and also perform a functional role of a local CA (Certification Authority) inside the Jini system. As shown in Figure 2, AAS (Authentication-Authorization Server) is a Jini-based application service that executes the functions for this purpose.

When clients and service providers join the lookup server, the lookup server offers their service proxy and the AAS proxy as default service proxy. Therefore, they can obtain an
access key to the lookup server from AAS through the AAS proxy, and then they can have a right to discovering and registering services that they want. In order to get access keys to various services from AAS, clients and service providers subscribe to AAS with their ID and public key and obtain signed public key of them with the private key of AAS. In particular, service providers additionally obtain the public key of CA and signed public key of AAS with the private key of CA as a public certification of the public key of AAS. These data are stored in a secure manner as a single local object in the server and client nodes.

Our proposed architecture can be used as a basic provision for building the secure service-oriented application based on DSNs, especially in the IoT environment based on DSN publishing and discovering the security context-dependent services [4].

4.2. Process of Improved Proxy Preparation

4.2.1. Exchange of Public Key Certificates. The proxy preparation is ultimately a process to prepare a stub connecting the client and the server in a secure way. In Jini security framework, after a stub is prepared, an authentication process is followed to authenticate the subject that uses the stub. This authentication process is the same as the existing TLS/SSL communication process based on X.509 certificate. In other words, this means that a certain connection delay occurs while the certification of subject's identity and their credential is being required in order to mutually authenticate subjects substantially using the stub. In addition, to provide the integrity and confidentiality of sessions, delay time for secret sharing is added to the cryptographic computation. We propose a method of exchanging public key certificates while the client and the server are preparing a stub. In our method, the service provider configures a service proxy to be added his own public key certificate object when exporting their service proxies. The public key certificate in server-side contains a 4-tuple data as follows:

\[
\text{PublicKe}_\text{Certificate} \leftarrow (EK_{\text{Server}}(K_{\text{Server}}^+), \\
EK_{\text{AAS}}(K_{\text{AAS}}^+), EK_{\text{CA}}(K_{\text{AAS}}^+), K_{\text{CA}}^+),
\]

where \(K_{\text{AAS}}^+\) indicates the public key of AAS, \(EK_{\text{AAS}}\) (parameter) indicates signing the parameter with the private key of AAS, and \(K_{\text{CA}}^+\) is the public key of CA that is used for the public certification of \(K_{\text{AAS}}^+\).

This public key certificate forms a certification chain that can be used to authenticate or trace the subject of the server-side stub in sensor service based on DSNs [19]. Before exporting a service proxy, the application that adds additional data for access control to this public key certificate can provide more solutions to reduce time cost for secure communication such as [22–24].

The server-side configuration that we propose is based on Jini configuration mechanism as shown in Algorithms 1 and 2.

The client obtains bootstrap proxy object and the public key certificate of server from the downloaded service proxy. The public key certificate can be used to authenticate the subject of server-side stub and the bootstrap proxy object can be used to request TrustVerifier object through getVerifier() in it.
In the implementation of getVerifier() of net.jini.security.proxytrust.ProxyTrust interface, we propose a method of delivering clients’ public key certificate and the nonce value for session encryption (or decryption) to the remote service implementation (server).

The client prepares a context object that contains a public-key certificate and a nonce value for session encryption (or decryption). The public-key certificate in client-side contains data that the server is able to certificate the identity and credential of the client. The context object can be obtained in the following way:

\[
\text{Context} \leftarrow \text{EK}_{\text{Server}}\left(K^+_{\text{Client}}, K^+_{\text{Client}}\right), \\
K^+_{\text{AAS}}\left(K^+_{\text{Client}}, \text{nonce}_{\text{client}}\right).
\] (2)

After the client authenticates the subject of service provider through the certification chain in public key certificate of server, the prepared context object is delivered to the remote service implementation (i.e., server) through the invocation of getVerifier(context), where, nonce_{client} indicates random values that the client generates.

4.2.2. Secret Sharing for Secure Communication. When the server receives an invocation of getVerifier(context), it authenticates the client by obtaining \(K^+_{\text{Client}}\) from the context object. At the same time, it obtains a nonce value as a secret for session encryption (or decryption) as well.

The server delivers TrustVerifier object and values of \(K^+_{\text{Client}}, \text{nonce}_{\text{Client}}\) \text{nonce}_{\text{Server}} to the client as a response. In this manner, the client can verify the trust of service proxy codes and share session secret keys with the server at the same time.

The provision of confidentiality through shared secret keys will subsequently lead to a fast response to encryption and decryption at the end point that connects the server with the client. Confirming the integrity of messages is conducted through HMAC algorithm.

5. Experiments and Discussions

This section discusses the experimental method and result to show the effect of our approach.

5.1. Experimental Environments and Methods. We established a single subnet composed of 4 PCs with a USB type sensor as a single Jini IoT system. Each PC was used to host Jini services or client software, depending on an experiment. For the development of software, we used JDK SE8 and Apache River release 2.2.2 including jini security API.

For this experiment, we implemented a simple echo service. When a certain-sized message was sent by the client, we measured the turn-around time. At this point, the turn-around times measured by the size of each message are independent measurement results. That is, every time when connection was attempted, it repeated the same process as a measurement of time taken for the client to get connected to the initial service and response.

| Task            | Time Costs | Latency (ms) |
|-----------------|------------|--------------|
| Proxy Preparation | 4,320       | 4,983        |
| Preparation of Session Key including Mutual Authentication | 3,450 | 2 |
| Session Encryption | 23         | 23           |

5.2. Experimental Results

5.2.1. Improvement of Turn-Around Time as a Service Delivery Time. In the standard Jini system, till the client accesses the server and obtains a response time expense should be taken for four steps: proxy preparation, mutual authentication for communication subjects, creation of session key based on certification, and session encryption. As directed by our proposed method, Figure 3 shows results that the delay time could be reduced by about 2.6 times, in proportion to the size of messages, because authentication data about subjects of communication were exchanged in advance in the proxy preparation step. In the experimental results of Figure 3, the time cost for the proxy preparation task is not included. The standard Jini system pays a higher cost in latency for mutual authentication and preparation of session key than our system.

5.2.2. Latency Comparison for Secure Service Connection. Table 1 shows the comparison of the time cost in latency for which the client accesses the server and obtains a response. In Table 1, task 1 in our system has been increased by 15.3% compared with the standard Jini system. Because our system exchanges the context object including data for mutual authentication and nonce value for session key share, time.
for task 2 based on exchanged data can be saved in data processing level without communication overhead. Because task 2 in our system is moved into task 1, it does not contribute to latency.

6. Conclusion

Jini security framework maintained in Apache River project has been reinforced with security mechanism to respond against mobile codes in the existing Java security models, in order to satisfy requirements for the security of Jini system, which can be vulnerable in the ubiquitous environment. We proposed a solution to reduce overheads causing unnecessary connection delay in every initial connection to service by completing the preparation of mutual authentication data and session key while on proxy preparation task. Through the improvement of proxy preparation, we obtained experimental results that could reduce connection delay by about 2.6 times, in proportion to the size of messages. Our work presents an approach to reduce the additional overhead that would be paid for by applying the Jini security framework to the development of sensor application based on Jini IoT.

A downloaded proxy keeping connections to the server can make use of the service within a lease time given by Lookup server. Thus, if the lease time is expired, there is no need to keep the expired proxy. One of the goals of our future research is to develop a scheme to manage the expired proxies among downloaded proxies to respond to security threats exploiting unmanaged proxies.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] R. Sugihara and R. K. Gupta, “Programming models for sensor networks: a survey,” ACM Transactions on Sensor Networks, vol. 4, no. 2, article 8, 2008.

[2] A. Pereira, N. Costa, and C. Serôdio, “Peer-to-peer Jini for truly service-oriented WSNs,” International Journal of Distributed Sensor Networks, vol. 2011, Article ID 616838, 13 pages, 2011.

[3] M. Weiser and S. Brown, “The coming age of calm technology,” Tech. Rep., Xerox PARC, 1996.

[4] N. Ibrahim, M. Mohammad, and V. Alagar, “Publishing and discovering context-dependent services,” Human-Centric Computing and Information Sciences, vol. 3, article 1, 2013.

[5] N. Mohamed and J. Al-Jaroodi, “A survey on service-oriented middleware for wireless sensor networks,” Service Oriented Computing and Applications, vol. 5, no. 2, pp. 71–85, 2011.

[6] E. Álvarez-López and J. A. García-Macias, “TinySOA: a service-oriented architecture for wireless sensor networks,” Service Oriented Computing and Applications, vol. 3, no. 2, pp. 99–108, 2009.

[7] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, “A service-oriented architecture for QoS configuration and management of wireless sensor networks,” in Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '10), Bilbao, Spain, September 2010.

[8] I. Amundson, M. Kushwaha, X. Koutsoukos, S. Neema, and J. Sztipanovits, “OASIS: a service-oriented middleware for pervasive ambient-aware sensor networks,” Pervasive and Mobile Computing Journal on Middleware for Pervasive Computing, 2006.

[9] M. P. de Vos, Integration of WSN with a SOA [M.S. thesis], Department of Mathematical and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2009.

[10] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, “A service-oriented architecture for QoS configuration and management of Wireless Sensor Networks,” in Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '10), pp. 1–8, Bilbao, Spain, September 2010.

[11] J. King, R. Bose, S. Pickles, A. Helal, S. Ploeg, and J. Russo, “Atlas: a service oriented sensor platform,” in Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN ’06), pp. 630–638, Tampa, Fla, USA, November 2006.

[12] J. Waldo, “The Jini architecture for network-centric computing,” Communications of the ACM, vol. 42, no. 7, pp. 76–82, 1999.

[13] M. Bertocco, S. Cappellazzo, C. Narduzzi, and M. Parvis, “A distributed sensor network based on Jini technology,” in Proceedings of the IEEE International Symposium on Virtual and Intelligent Measurement Systems (VIMS ’02), pp. 68–71, Girdwood, Alaska, USA, 2002.

[14] Apache Software Foundation, “Apache Jini Specifications v2.1.2,” Published Specification, http://river.apache.org/doc/spec-index.html.

[15] Apache Software Foundation, “Apache River User Guide,” http://river.apache.org/user-guide-basic-river-services.html.

[16] F. Sommers, Jini Starter Kit 2.0 Tightens Jini’s Security Framework, IEEE Computer Society Press, Los Alamitos, Calif, USA, 2003.

[17] M. Schönefeld, “Hunting flaws in JDK,” in Proceedings of the Blackhat Euope, Amersdam, The Netherlands, May 2003.

[18] P. Hasselmeyer, R. Kehr, and M. Voß, “Trade-offs in a secure jini service architecture,” in Trends in Distributed Systems: Towards a Universal Service Market, vol. 1890 of Lecture Notes in Computer Science, pp. 190–201, Springer, Berlin, Germany, 2000.

[19] P. Eronen and P. Nikander, “Decentralized Jini security,” in Proceedings of the Network and Distributed System Security Symposium (NDSS ’01), pp. 161–172, San Diego, Calif, USA, February 2001.

[20] T. Schoch, O. Krone, and H. Federrath, “Making jini secure,” in Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4 ’01), pp. 276–286, Dallas, Tex, USA, November 2001.

[21] Oracle, “Java Secure Socket Extension(JSSE) Reference Guide for Java Platform Standard Edition 6,” http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html.

[22] C.-W. Chen, Y.-R. Tsai, and S.-J. Wang, “Cost-saving key agreement via secret sharing in two-party communication systems,” Journal of Convergence, vol. 3, no. 4, pp. 29–36, 2012.

[23] T.-T. Truong, M.-T. Tran, and A.-D. Duong, “Improvement of the more efficient & secure ID-based remote mutual authentication with key agreement scheme for mobile devices on ECC,” Journal of Convergence, vol. 3, no. 2, pp. 25–36, 2012.

[24] K. Peng, “A secure network for mobile wireless service,” Journal of Information Processing Systems, vol. 9, no. 2, pp. 247–258, 2013.
