Computationally Efficient PKI-Based Single Sign-On Protocol PKASSO for Mobile Devices

Ki-Woong Park, *Student Member*, *IEEE*, Sang Seok Lim, *Member*, *IEEE*, and Kyu Ho Park, *Member*, *IEEE*

Abstract—In an attempt to expand Public Key Infrastructure (PKI) usage to a ubiquitous and mobile computing environment, we found that the deployment of the PKI on a resource-constrained device such as an 8-bit microprocessor leads to user-obstructive latency or additional circuitry for the operations. To alleviate these limitations, we propose a new PKI-based authentication protocol and security infrastructure, namely, PKASSO, which is enhanced with the single sign-on and delegation technology that is used especially for mobile devices with restricted computation power. PKASSO offloads complex PKI operations from the mobile devices to the infrastructure so as to keep the hardware and software complexity of the devices as low as possible. In addition, even though a conventional delegation mechanism cannot support a nonrepudiation mechanism against malicious user behavior, PKASSO can provide such a mechanism by devising a referee server that, on one hand, generates binding information between a device and authentication messages and, on the other hand, retains the information in its local storage for future accusation. We present the detailed design and performance evaluation of PKASSO and offer a protocol analysis in terms of user authentication latency and the completeness of the protocol. According to the performance evaluation, the authentication latency of our infrastructure (which averages 0.082 second) is much shorter than the authentication latency of a conventional PKI-based authentication latency (which averages 5.01 seconds).

Index Terms—Authentication, access controls, public key cryptosystems.

1 Introduction

In the provision of services in a ubiquitous environment, pervasive devices frequently communicate with other unknown devices, thereby exposing themselves to an unfortified insecure environment. Security in a ubiquitous environment is therefore of immense importance and the issue needs to be addressed to secure the privacy and confidentiality of users who live in such an environment [1]. In our ubiquitous environment, we have been developing a wearable computer, called the Ubiquitous Fashionable Computer (UFC) [2] and, in its interoperable computing environments, we deployed various devices such as U-Kiosks, U-Print, and ZigBee-enabled appliances to provide users with ubiquitous services, primarily for a university campus [3].

As a fundamental way of enforcing the security of a ubiquitous environment, authentication and authorization stand out as the two most widely used mechanisms. A full realization of trustworthy ubiquitous services based on authentication and authorization in a ubiquitous computing environment requires deep understanding of the performance characteristics and the communication patterns of the mobile devices since those devices are usually endowed with limited computation power. In particular, due to the anonymity and mobility of users in a ubiquitous environment, the pervasive devices placed to provide services should always check the identity of the mobile devices and vice versa, necessitating mutual authentication between them. Thus, fundamental security operations such as authentication, digital signatures, nonrepudiation, and secure key distribution should be provided as essential services by a security infrastructure to ensure the high security of the services [4].

From a security aspect, the Public Key Infrastructure (PKI) is generally considered as the most appropriate solution for the requirements. However, the computational complexity of PKI causes high deployment costs and a high operational overhead because asymmetric key operations of PKI need to be performed on mobile devices. In addition, the frequency of user authentication requests increases rapidly in proportion to the number of the service devices that require mutual authentication [5]. In a mobile computing environment, the performance implications of protocol design are often accentuated by limitations in the capacity of the mobile processor and the wireless network. The resources required to perform asymmetric key operations and to transmit large messages may result in unacceptable performance results and intolerable user authentication response times.

By thoroughly investigating our ubiquitous environment and conventional security systems, we identified two critical limitations of a conventional PKI-based security system. The limitations and our approach are described as follows:

1. **Obstructive authentication latency in a mobile device with restricted computation power.** There is a challenge with clients where mobile devices with

- *K.-W. Park and K.H. Park are with the Electrical Engineering Department, KAIST, 373-1 Gusung-dong Yousung-gu, Daejon, Korea 305-701. E-mail: woongbak@core.kaist.ac.kr, kpark@ee.kaist.ac.kr.*
- *S.S. Lim is with Samsung Electronics Co. Ltd., 416 Maetan-dong Youngtong-gu, Suwon, Korea 443-370. E-mail: sslim@core.kaist.ac.kr.*

Manuscript received 28 June 2007; revised 20 Dec. 2007; accepted 12 Feb. 2008; published online 20 Feb. 2008.
Recommended for acceptance by S. Nikoletseas.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-06-0265.
Digital Object Identifier no. 10.1109/TC.2008.36.

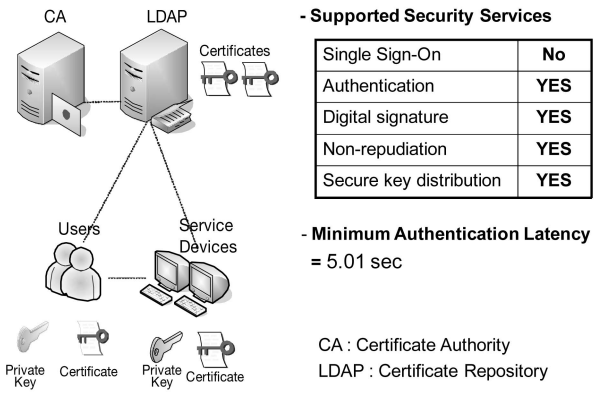


Fig. 1. The conventional PKI architecture and a brief overview of the characteristics.

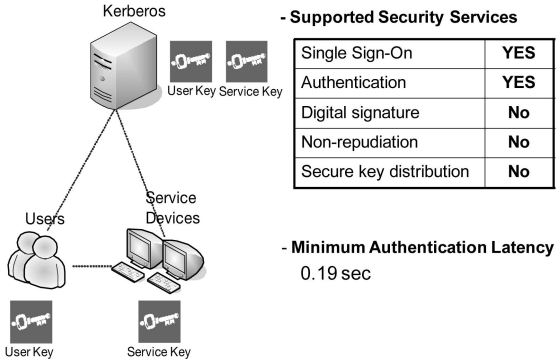


Fig. 2. The architecture of a conventional security infrastructure based on Kerberos and its characteristics.

restricted computation power demand frequent and dynamic authentications over PKI. To address this latency problem, we propose a security infrastructure that is based on PKI and a single-sign-on (SSO) protocol. The proposed security infrastructure provides users with a seamless and secure system of authentication and key distribution. In the proposed infrastructure, we devised a delegation server that offloads complex PKI operations from mobile devices to the infrastructure so as to keep the hardware and software complexity of the devices as low as possible. By using the proposed authentication mechanism, we can also achieve a more secure mechanism in line with newly emerging security policy requirements without replacing hardware or software components of devices.

2. **Support for digital signature and nonrepudiation.** Even though a digital signature mechanism and a nonrepudiation mechanism are essential functions in security applications, a conventional delegation mechanism cannot support these two mechanisms against malicious user behavior. Because user authentication transactions are already delegated, an authenticator cannot prove the fact that the authentication transactions are really intended by users. Our proposed protocol, named PKASSO (Public Key-based A^3-providing Single Sign On), and the security infrastructure can provide the two mechanisms by means of a referee server that generates binding information between a device and authentication messages [6].

The remainder of this paper is organized as follows: In Section 2, we discuss relevant works. In Section 3, we present the overall system design and components of the proposed security infrastructure. In Section 4, we illustrate the proposed SSO protocol. In Section 5, we evaluate the performance of PKASSO. Finally, in Section 6, we present our conclusions.

2 Related Works

To meet the demands of a ubiquitous and mobile environment, researchers have endeavored to facilitate the deployment of a security infrastructure for e-commerce, particularly an infrastructure that can verify the authenticity of communicating parties and establish trust among devices over the air. In this section, we briefly discuss the experimental results as we evaluate the preexisting security infrastructures in terms of security level and authentication latency. More comprehensive evaluations of the experimental results are described in Section 5.

2.1 PKI

PKI, which is a representative security infrastructure based on an asymmetric key, is generally considered to be the most appropriate solution for e-commerce and mutual authentication due to its digital signature and nonrepudiation features [7]. The organization of a conventional PKI and its characteristics in terms of security level and authentication latency are illustrated in Fig. 1. Even though PKI provides full security features, including authentication, digital signature, nonrepudiation, and secure key distribution, it has a severe drawback when used by a diminutive device with restricted computation power, that is, the user service latency, which is mainly determined by the authentication latency, is exacerbated by the restricted resources of the device and by the extremely high complexity of RSA operations for encryption, decryption, and certificate validation in PKI. The estimated minimum authentication latency in our experimental environment is about 5.01 seconds on the 8 bit microprocessor [8] deployed mainly in a wireless communication module.

2.2 Kerberos-Assisted Authentication

Fig. 2 shows the architecture and characteristics of a conventional authentication system based on a symmetric cryptography called Kerberos [9]. Kerberos achieves a significantly reduced authentication latency (minimum of 0.19 second) over PKI (minimum of 5.01 seconds) by virtue of the lesser complexity of the symmetric key operations and the SSO mechanism. Kerberos works on the basis of tickets, namely, a Ticket Grant Ticket (TGT) and a Session Grant Ticket (SGT), which prove the identity of users. Whenever a user attempts to connect to a specific service, a TGT and an SGT are required for Kerberos authentication. Once a user acquires a TGT from the Kerberos server through the initial user authentication, the user can get SGTs which enable the user to receive a service several times without additional authentication for a specific period.

Kaman [9] studied how the Kerberos protocol can be deployed in mobile ad hoc networks. He presented a secure key exchange scheme for use in ad hoc networks. The

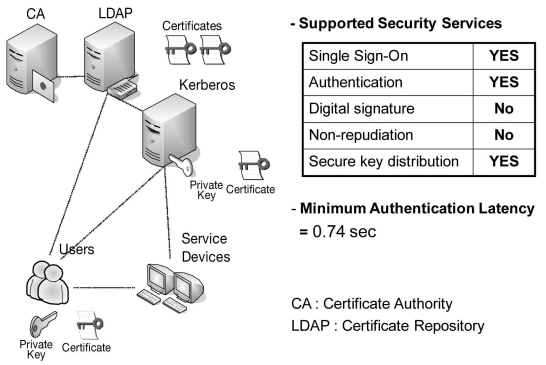


Fig. 3. Overall M-PKINIT architecture and a brief overview of the characteristics.

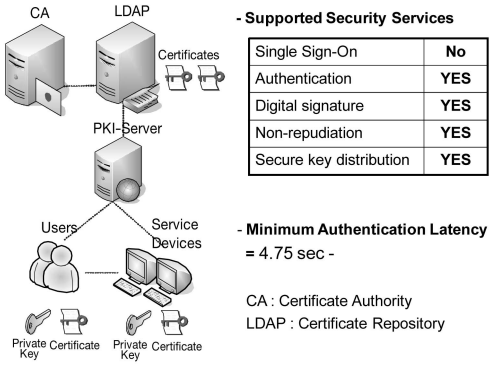


Fig. 4. NSI architecture and a brief overview of the characteristics.

scheme is based on the Kerberos protocol and measures factors such as replication and elections to ensure maximum connectivity between clients and servers. This scheme cannot, however, provide a digital signature, nonrepudiation, and secure key distribution for unknown devices, even though these functions are essential in security applications. These functions are precluded, because Kerberos uses an authentication mechanism based on a symmetric cryptography with a preshared secret key.

2.3 PKINIT and M-PKINIT

PKINIT is an extension of the Kerberos protocol so as to enable public key-based authentication between a user and a Key Distribution Center (KDC) instead of using symmetric key-based authentication. In this approach, a Kerberos server collaborates with the PKI entities, CA and LDAP directory server [10]. In addition, the PKINIT protocol aims at enhancing the security of the Kerberos protocol by using a PKI-based authentication to get a TGT by using a smaller number of public key operations than conventional PKI-based authentication.

The M-PKINIT protocol is a lighter version of the PKINIT which was proposed in an attempt to reduce the significant overhead of the authentication operation and communication whenever public key protocol operations are invoked in a mobile device [11]. Fig. 3 shows the organization and characteristics of the M-PKINIT protocol. This scheme can provide a secure key distribution mechanism over the asymmetric key operation, SSO through Kerberos, and it has a shorter authentication latency (minimum of 0.74 second) than the PKI (minimum of 5.01 seconds). However, it cannot provide a digital signature and nonrepudiation because each authentication for a service device is accomplished by Kerberos-based authentication. The M-PKINIT protocol requires performing public and private key operations on a user's device whenever the user moves to another Kerberos realm to get a TGT.

2.4 NSI

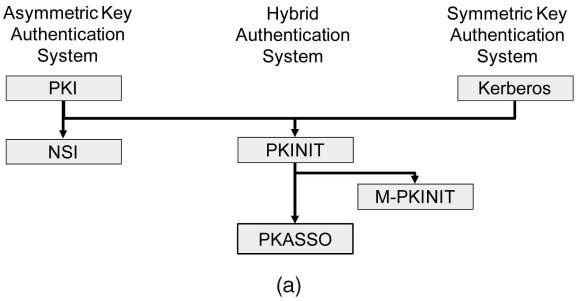
The NSI introduced a PKI server that is responsible for searching and verifying certificates on behalf of a mobile device or client [12]. Fig. 4 shows the architecture and characteristics of the NSI. The PKI server provides a set of simple and abstract APIs (PKI Client API) that hide most of the complex PKI operations from a client, thereby minimizing the hardware and software complexity for a client. To minimize the operational complexity for the client and server, we adopted a mechanism, called delegation [13], which has been actively studied and used in the research area of grid computing. The PKI server and the delegation mechanism can both minimize the PKI-related computation overhead of a client by offloading the complex operations to a powerful infrastructure server. Although the NSI provides all of the security functionality of the PKI, the authentication latency of the NSI (minimum of 4.75 seconds) fails to meet our system requirement with respect to authentication latency. The major reason for that failure is because the asymmetric key operations are performed on the user's mobile device.

2.5 Summary

Fig. 5 shows the technical advancement flow of the aforementioned works, as well as their security functionalities and authentication latency. The last row in Fig. 5b clarifies our PKASSO's design goal with respect to security functionalities to be achieved and its resulting authentication latency.

Even though PKI has attained consensus that it can enable high security by providing the four essential security functionalities, except for SSO, it comes at the price that the PKI-based authentication requires many more CPU cycles than the Kerberos authentication system based on a symmetric cryptography. Consequently, when PKI is used in a mobile and ubiquitous environment, the high computational complexity results in high deployment costs and operation overhead since those operations are necessarily performed on a mobile device with computation power constraints. As compared to PKI, Kerberos and M-PKINIT require relatively low computational complexity and provide SSO capability, leading to a significantly reduced authentication latency of 0.19 and 0.74 second, respectively. Kerberos, however, is incapable of providing digital signature, nonrepudiation, and secure key distribution due to its nature of asymmetric key operations and the nonexistence of CA, a third-party security mediator. M-PKINIT cannot provide a digital signature and nonrepudiation, though they can provide secure key distribution by initial authentication over PKI. Although NSI can provide all of the security functionality of PKI and reduce the PKI operation complexity on a mobile device, it still suffers from an intolerable authentication latency. In addition, NSI is not designed to provide SSO capability.

To alleviate these limitations, we propose a new PKI-based authentication protocol and security infrastructure, namely, PKASSO. To harness the full power of PKI, even in a mobile environment, NSI and PKASSO take a different



(a)

System	Security Service					Latency
	Single Sign-On	Authentication	Digital Signature	Non-Repudiation	Secure Key Distribution	
PKI	No	Yes	Yes	Yes	Yes	5.01sec
Kerberos	Yes	Yes	No	No	No	0.19sec
PKINIT	Yes	Yes	No	No	Yes	1.21sec
M-PKINIT	Yes	Yes	No	No	Yes	0.74sec
NSI	No	Yes	Yes	Yes	Yes	4.75sec
PKASSO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**0.082sec**

(b)

Fig. 5. Summary of relevant works. Experimental results in terms of authentication latency are described in Section 5. (a) Evolution of relevant works. (b) Security functionality and authentication latency of relevant works.

stance by making an effort to reduce the number of PKI operations or the complexity of asymmetric key operations, as shown in Fig. 6. The main goal of NSI is to offload the certificate validation phase (searching and verifying a certificate through a lengthy validation chain) out of a mobile device. In our experiment, as shown in Fig. 6, the overall authentication latency of NSI was reduced to 4.75 seconds. On the other hand, PKASSO aims at offloading the mutual authentication phase (mainly asymmetric key operations) and the certificate validation phase from a mobile device to the infrastructure. In terms of authentication latency, PKASSO itself is broken into two phases: the delegation phase and the authentication phase in Fig. 6. Upon entering the security infrastructure for the first time, the mobile device needs to perform asymmetric key operations for delegation (5.19 seconds). Once the delegation is successfully completed, from the second time, the authentication latency is reduced to 0.082 second.

3 Design of the PKASSO Security Infrastructure

3.1 The PKASSO Design Philosophy

As explained in Section 2, we designed the PKASSO security infrastructure to solve the limitations of a conventional authentication system. Our PKASSO design philosophy is based on the following objectives:

- **Nonobstructive authentication latency.** The authentication latency of PKASSO should be bounded within a certain amount of time, particularly in terms of human perception and the type of services for which it will be used.
- **Equivalent security level as the PKI.** PKASSO should provide the same security features, including authentication, digital signature, nonrepudiation, and secure key distribution, within a nonobstructive authentication latency.

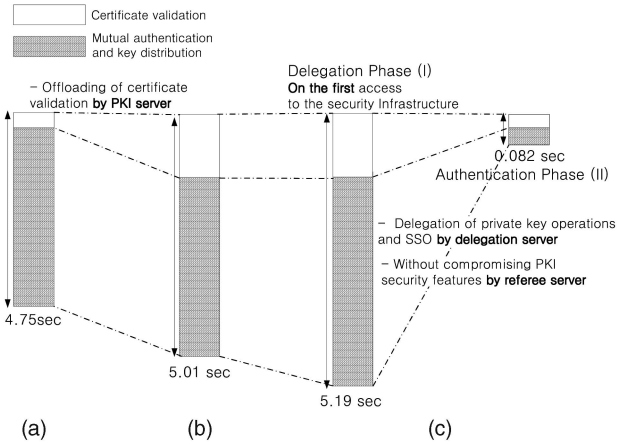


Fig. 6. Comparison of authentication latency for PKIX, NSI, and PKASSO.

3.2 Overall System Design

We incorporated the PKINIT protocol into our system as an underlying security infrastructure in an attempt to provide a secure key distribution and a way of efficiently and cost effectively managing a large number of devices and sensors in a ubiquitous environment. As described in Section 2, the PKINIT protocol is an extension of the Kerberos protocol so as to enable public key-based authentication between a user and KDC instead of using the symmetric key-based authentication. However, the PKINIT protocol has two drawbacks: 1) It cannot provide a digital signature and nonrepudiation because each authentication for a service device is accomplished by Kerberos-based authentication and 2) it has an obstructive authentication latency (minimum of 0.74 second) because it requires public and private key operations on the user's device whenever the user moves to another Kerberos realm to get a TGT. To overcome these drawbacks, we based the design of our security infrastructure and protocol on two principles:

- **A computationally efficient PKI-based SSO protocol.** The requirement of users to sign on for each service device on each occasion will severely hamper the usability of a diminutive security device with limited computation power. The widely used SSO technology can greatly relieve the poor usability problem by obviating the need for repeated sign-on procedures; hence, we adopted the SSO technology into our underlying security infrastructure. To provide the SSO technology and nonobstructive authentication latency, we propose a computationally efficient PKI-based SSO protocol that is based on a delegation mechanism that uses a proxy certificate [13]. The proposed protocol also provides identical security functionalities as in PKI.
- **A delegation server and a referee server.** For our SSO protocol, we devised a delegation server and a referee server. The delegation server is responsible for performing prohibitively expensive PKI operations on behalf of a diminutive security device to minimize the computational overhead of the security device. The referee server, which is designed to provide a computationally efficient digital signature and nonrepudiation, generates binding information

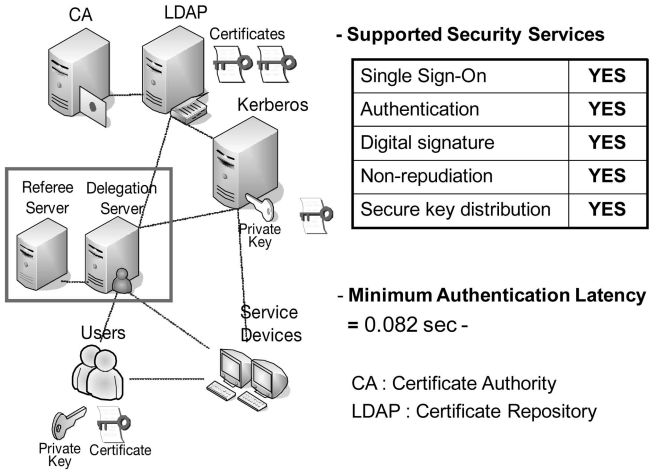


Fig. 7. The proposed security infrastructure PKASSO.

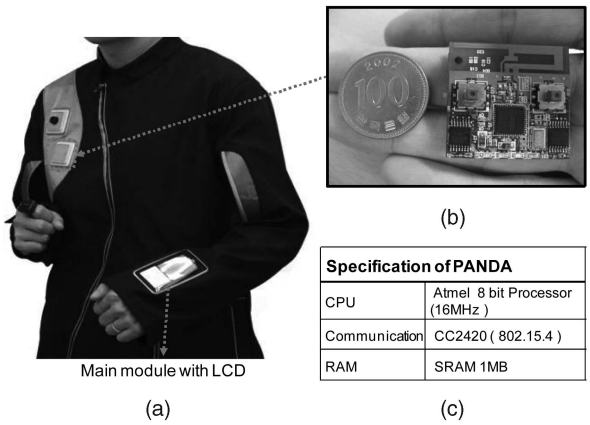


Fig. 8. (a) UFC [2] and UFC modules attached on UFC. The UFC modules (Camera module and PANDA) can be attached and detached easily on UFC, allowing users to construct one's own UFC platform. (b) Developed PANDA. (c) Specification of the PANDA [14].

between security devices and authentication messages and retains the information in its local storage for future accusation.

3.3 The Proposed PKASSO Security Infrastructure

Fig. 7 shows the overall architecture of our PKASSO security infrastructure. The five major components of our architecture are listed as follows:

- The **PKINIT protocol**, which is considered as a promising foundation of PKASSO, consists of a CA, an LDAP, and a Kerberos server, as described in Section 2.3.
- A **user** is a mobile entity that receives the provided services in our ubiquitous security environment. To utilize services such as authentication, authorization, and accounting in the environment, each user should carry a diminutive security device called a PANDA, which is a type of a smart card equipped with ZigBee-based low-power intercommunication capability and location-sensing capability [15]. Fig. 8 shows our implementation of the PANDA and the UFC.
- A **service device** permeates the surroundings for the provision of services. The service device in our environment has a ZigBee communication module to interact with users and to authenticate the identity of users.
- A **delegation server** is designed to offload complex PKI-related operations from a user to the infrastructure, making it possible to develop users' security device PANDA with cheap simple hardware. The delegation server also maintains all of the proxy certificates that contain the private keys and public keys that are delegated and signed by a user. Upon entering the security infrastructure for the first time, the user delegates the user authentication operations to the delegation server by following RFC3820 [13]. The delegation server subsequently takes over all of the authentication operations until the users' proxy certificate expires.
- A **referee server** provides a nonrepudiation mechanism for combating malicious user behavior. The nonrepudiation mechanism takes effect as long as the user uses its own private key in an authentication process. However, after delegating its operations to the delegation server, the user no longer uses its private key as a means of halting the provision of the nonrepudiation mechanism. We therefore devised a referee server so that, even during the delegation process, we could bring the mechanism back into our system. The referee server investigates all authentication messages, generates binding information on the fly between a user and the authentication messages, and retains the information signed by the referee server's private key in its local storage for future accusation. Due to the referee server, our proposed security infrastructure can provide a symmetric key-based nonrepudiation mechanism that is computationally efficient on the user's mobile devices. More specific details of the nonrepudiation mechanism are described in Section 4.

3.4 Overall Authentication Process of the PKASSO

Fig. 9 shows the overall process of authentication on the basis of our security infrastructure. The proposed delegation server and the referee server make it possible to authenticate users with only three symmetric key operations on the user's device after the delegation, thereby providing the following level of security, which is identical to the security of PKI:

1. The service device sends a challenge message to a user who intends to receive a service.
2. The user generates an authentication request message (with two symmetric key operations) and sends it to the delegation server.
3. The delegation server performs transactions for verification and authentication with the referee server and the PKINIT protocol on behalf of the user.
4. By using a received authentication message from the PKINIT protocol, the delegation server makes a response message and transmits it to the service device.
5. The authentication is completed on the arrival of a confirming message from the user.

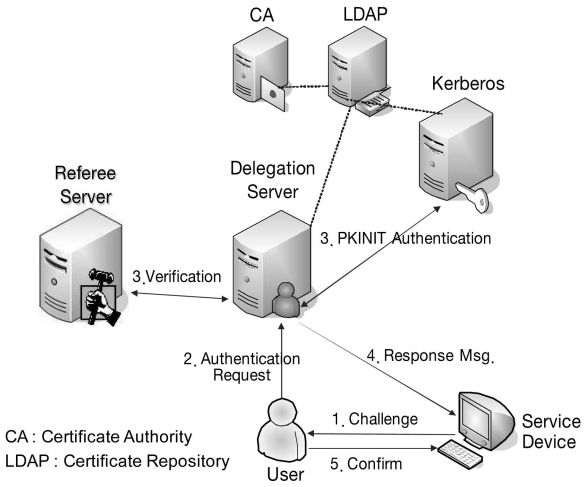


Fig. 9. Authentication flow description with the delegation server.

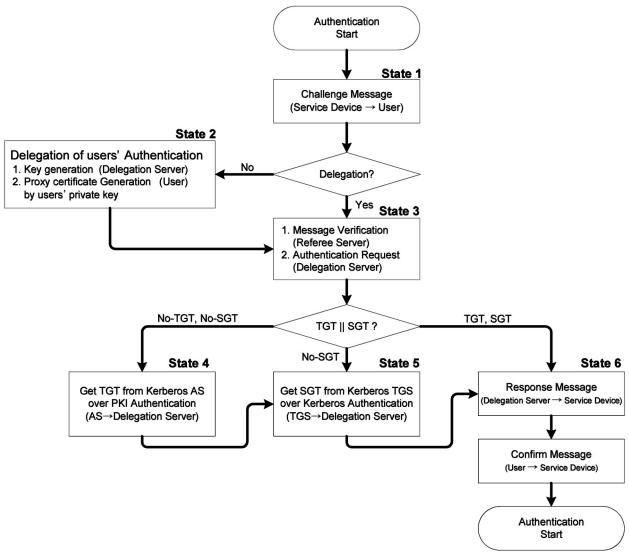


Fig. 10. Flow diagram of the proposed SSO protocol.

4 Proposed SSO Protocol

Deliberating on our system design philosophy, we made our best effort to streamline the computation of a mobile device with restricted computation power and to minimize the communication overhead per authentication in a mobile device. Our novel challenge message, which is generated by a hash function and a cross-distributed key among the entities, can facilitate authentications with optimal computation and communication overheads without compromising any of the system's PKI security standards.

4.1 Flow Diagram of the SSO Protocol

Fig. 10 shows a flow diagram of the proposed SSO protocol. The protocol consists of six states and the protocol safety is verified in Section 4.4:

- **State 1.** When a user intends to receive a service from a service device, the user accepts a beaconing challenge message from the service device.
- **State 2.** On the first occasion when a user accesses the security infrastructure, the user and the delegation server perform a mutual authentication based on PKI. The user then delegates the user authentication operations to the delegation server. For this purpose, a public-private key pair is generated by the delegation server and the public key is subsequently transmitted to the user. Upon the arrival of the public key, the user generates a proxy certificate containing the public key and signs the proxy certificate with the private key. Last, the user sends the proxy certificate back to the delegation server. Through the delegation mechanism, three keys are shared exclusively between the user, the delegation server, and the referee server. Throughout the public key operation, the user shares two keys: one with the delegation server and one with the referee server. The delegation server also shares one key with the referee server and one with the user.
- **State 3.** Once the delegation is successfully completed, *State 2* is skipped until the corresponding proxy certificate either expires or is revoked. The user encrypts the received challenge message by using the AES twice; the user then transmits the encrypted message to the delegation server. Upon receiving the encrypted challenge message from the user, the delegation server asks the referee server to verify the message.
- **States 4-6.** Upon reception of the verification result from the referee server, the delegation server operates the PKINIT authentication to get a *TGT* over the PKI (*State 4*) and an *SGT* over the Kerberos (*State 5*). Finally, the delegation server generates and sends a response message to the service device and the user sends a confirm message to the service device. After the validation check of the service device, the authentication operation is terminated (*State 6*).

4.2 Description of the SSO Protocol

As previously mentioned, the proposed protocol can provide seamless authentication after the delegation. Any user who accesses the security infrastructure for the first time is asked to delegate the user authentication. Hence, the initial authentication takes longer than delegated authentications. On the other hand, the authentication latency can be shortened drastically after the completion of the delegation because the user can be authenticated by using a much simpler type of symmetric cryptography. Thus, the authentication flow is changed as follows:

- The first access to the security infrastructure goes through all of the states, i.e., from State 1 to State 6.
- After the delegation, the authentication goes through State 1 and then moves to States 3-6.
- If the delegation server has already obtained a TGT, it goes through States 1, 3, 5, and 6.
- If the delegation server has already obtained a TGT and an SGT, it goes through States 1, 3 and 6.

In this section, we describe the authentication and the nonrepudiation mechanisms of the proposed SSO protocol.

TABLE 1
Notations of the Entities and Messages

Definition of the Entity Symbols
$Alice$: User
Bob: Service Device
$Delg$: Delegation Server
$Refe$: Referee Server
$Kerb\_AS$: Kerberos Authentication Server(AS)
$Kerb\_TG$: Kerberos Ticket Grant Server(TGS)

Definition of the Message Symbols
ID_x: ID of X
PU_x: Public Key of X
PR_x: Private Key of X
$PU_{x,y}$: Delegated Public Key of X by Y
$CR_{x,y}$: Proxy Certificate of X signed by Y
$Nonce$: Generated random data against replay attacks
$K_{x,y}$: Symmetric key shared between X and Y
SN: Serial number

Denotation
$A \rightarrow B$: Message1
Message1 = $\alpha \mid\mid \beta$.
A: Sender, B: Receiver
Message1 contains two contexts that are α and β.

Table 1 describes the notations of the entities and messages that describe the proposed protocol.

4.2.1 State 1

In *State 1*, as described in Fig. 11, a service device (*Bob*) keeps beaconing **Message 1-1**, which consists of the service ID. The service device also periodically sends a $Bob\_Capsule$ until a PANDA bearer (*Alice*) initiates an authentication process after receiving **Message 1-1**. The $Bob\_Capsule$ is the hashed value of two inputs: a unique serial number and a randomly generated nonce. The serial number is updated for each authentication transaction so that all of the serial numbers can be mapped uniquely to the user's ID and authentication transactions. Furthermore, the used seed value of the generated $Bob\_Capsule$ is unknown to the other entities.

4.2.2 State 2

The message flow of *State 2* is shown in Fig. 12. *State 2* is the phase for conducting a user delegation by generating a proxy certificate. If *Alice* has already delegated herself, the state transits to *State 3*. Otherwise, *Alice* needs to start the delegation process by entering *State 2*. To delegate authentication operations, *Alice* sends a delegation request message **Message 2-1**. In response, the delegation server generates a private-public key pair and sends the public key back to *Alice*, i.e., **Message 2-2**. *Alice* uses her private key to generate a proxy certificate containing the public key and she transmits the public key to the delegation server, i.e., **Message 2-3**. Next, the delegation server registers *Alice*'s identity and sends to the referee server the evidentiary data $K_{Refe,Alice}$, $K_{Delg,Refe}$, and $CR_{Delg,Alice}$ for nonrepudiation, i.e., **Message 2-4**. Last, the referee server grants a sequence

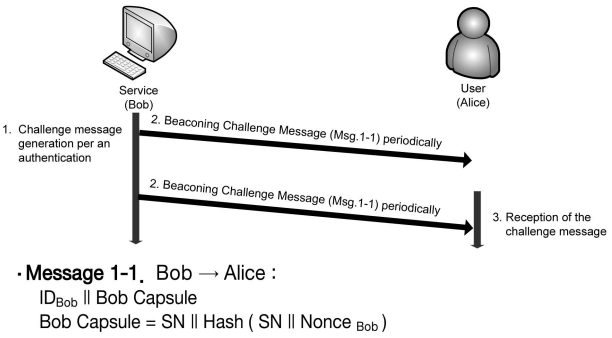


Fig. 11. *State 1*: message transaction of the proposed SSO protocol.

number Seq_{Alice}, which is uniquely mapped to the delegated user over **Message 2-5**.

4.2.3 State 3

State 3 is the phase of generating and sending an authentication request to a delegation server. Upon receiving a challenge message, i.e., **Message 1-1**, from the service device, *Alice* generates an authentication request message, i.e., **Message 3-1**, by combining the challenge message with a subkey (a symmetric key), which will be secretly shared with the service device *Bob*. *Alice* then sends **Message 3-1** to the delegation server. Subsequently, the delegation server generates **Message 3-2** as a proof that *Alice* requested an authentication for the service and then sends this message to the referee server for nonrepudiation. Upon the arrival of the message, the referee server checks the validity of the authentication request message and sends the result to the

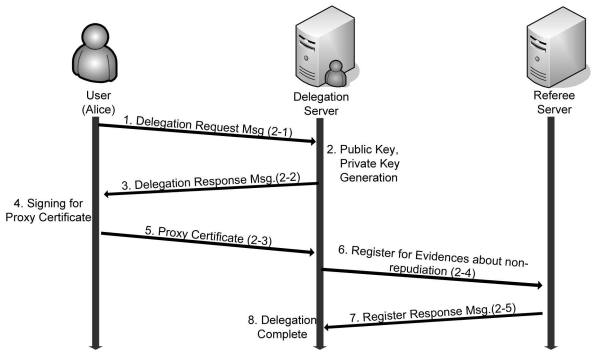
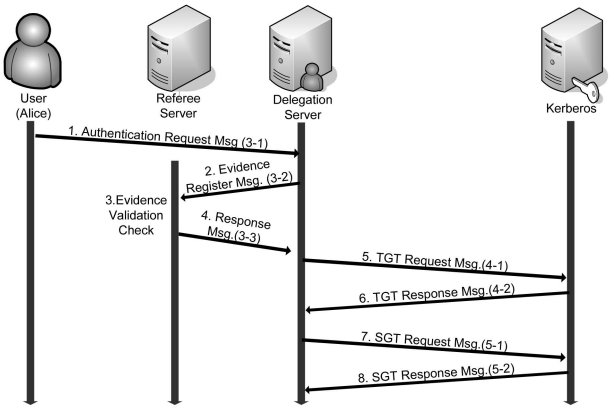


Fig. 12. State 2: the proposed delegation mechanism.



- **Message 3-1.** Alice → Delg :
  $ID_{Alice} \| E\{K_{Alice,Delg}, ID_{Alice} \| ID_{Bob} \| Bob\_Capsule \| Subkey_{Alice,Bob} \| E\{K_{Refe,Alice}, Bob\_Capsule\}\}$
- **Message 3-2.** Delg → Referee :
  $ID_{Delg} \| E\{K_{Delg,Refe}, ID_{Bob} \| Seq_{Alice} \| Nonce_{Refe} \| Bob\_Capsule \| E\{PR_{Alice,Delg}, E\{K_{Refe,Alice}, Bob\_Capsule\}\}\}$
- **Message 3-3.** Referee → Delg :
  $E\{K_{Delg,Refe}, OK \text{ or } BAD \| Nonce_{Refe}\}$
- **Message 4-1.** Delg → Kerb_AS :
  $E\{PU_{Kerb}, TGT\text{-}REQ \| CR_{Alice,Delg} \| E\{PR_{Delg}, CR_{Alice,Delg} \| K_{Kerb\_AS,Delg}\}\}$
- **Message 4-2.** Kerb_AS → Delg :
  $ID_{Alice} \| TGT_{Alice,TGS} \| E\{K_{Delg,}, K_{Kerb\_AS,Delg} \| Nonce_{TGT} \| Times\}$
- **Message 5-1.** Delg → Kerb_TGS :
  TGS_{Alice}-REQ
- **Message 5-2.** Kerb_TGS → Delg :
  TGS_{Alice}-REP

Fig. 13. Message flow diagram from *State 3* to *State 5*.

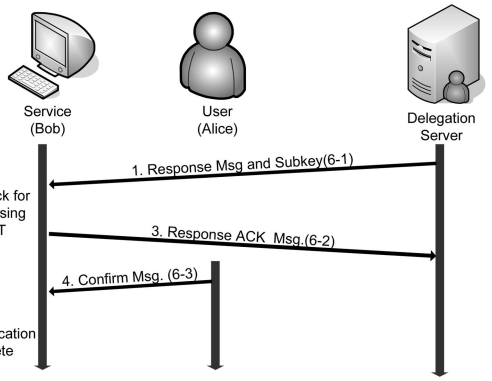
delegation server in **Message 3-3**. If the delegation server receives the *OK* message, the state moves to *State 4*.

4.2.4 State 4

The delegation server sends the *TGT* request message to the Kerberos server in case the delegation server does not have the previously issued *TGT* for *Alice*, i.e., **Message 4-1**. On the other hand, if the delegation server already holds the previously issued TGT for *Alice*, the protocol promptly moves to *State 5* without doing anything in this state. The Kerberos server responds to the *TGT* request message and sends the *TGT* to the delegation server in **Message 4-2**. Because each Kerberos server and the delegation server have their own certificates, they can authenticate each other by using an existing PKI.

4.2.5 State 5

If the delegation server obtains a *TGT* in *State 4*, the delegation server sends an *SGT* request message to the Kerberos server in **Message 5-1**. In response to the request message, the Kerberos server sends a new *SGT* for *Bob* to the delegation server in **Message 5-2**. If the delegation server gets the *SGT* in advance for the user authentication request, this state can be skipped and the protocol can proceed directly and immediately from *State 1* to *State 6*. Fig. 13 shows the message flows from *State 3* to *State 5*.



- **Message 6-1.** Delg → Bob :
  $SGT_{Alice} \| Nonce_{Delg} \| E\{K_{Delg,Bob}, ID_{Alice} \| Subkey_{Alice,Bob} \| Bob\_Capsule\}$
- **Message 6-2.** Bob → Delg :
  $E\{K_{Delg,Bob}, Subkey_{Alice,Bob} \| Nonce_{Bob}\}$
- **Message 6-3.** Alice → Bob :
  $ID_{Alice} \| E\{Subkey_{Alice,Bob}, Bob\_Capsule\}$

Fig. 14. Message flow diagram of *State 6*.

4.2.6 State 6

State 6 is the phase where the delegation server sends the final response message to the service device *Bob* in **Message 6-1**. *Bob* then checks the validity of the authentication by using the *Bob_Capsule* and shares the subkey generated by *Alice* in **Message 3-1**. After that, *Bob* sends the response message to the delegation server for mutual authentication in **Message 6-2**. Finally, the authentication is completed by *Alice*'s confirm message, i.e., **Message 6-3**. As a result, *Alice* and *Bob* can share the subkey after the authentication. Fig. 14 shows the message flow of *State 6*.

4.3 Computationally Efficient Nonrepudiation Mechanism

Our proposed protocol can provide nonrepudiation and a digital signature through the *Bob_Capsule*, which is generated among the entities by a hash function and distributed keys ($K_{Delg,Refe}$, $K_{Refe,Alice}$, and $K_{Alice,Delg}$). Upon entering the security infrastructure for the first time, user devices operate asymmetric key operations for delegation. Once the delegation is successfully completed, our protocol does not require any asymmetric key operations on the user devices; hence, PKASSO provides a level of security that is identical to that of PKI, as well as a minimized authentication latency.

This section elaborates upon how nonrepudiation can be provided in collaboration with the referee server. To achieve nonrepudiation, the referee server registers the delegation information of a user in *State 2* (the registration phase) and verifies the validity of the user's message for each authentication transaction in *State 3* (the verification phase). These transactions for nonrepudiation require no additional cryptography operations and communication overhead on the user devices.

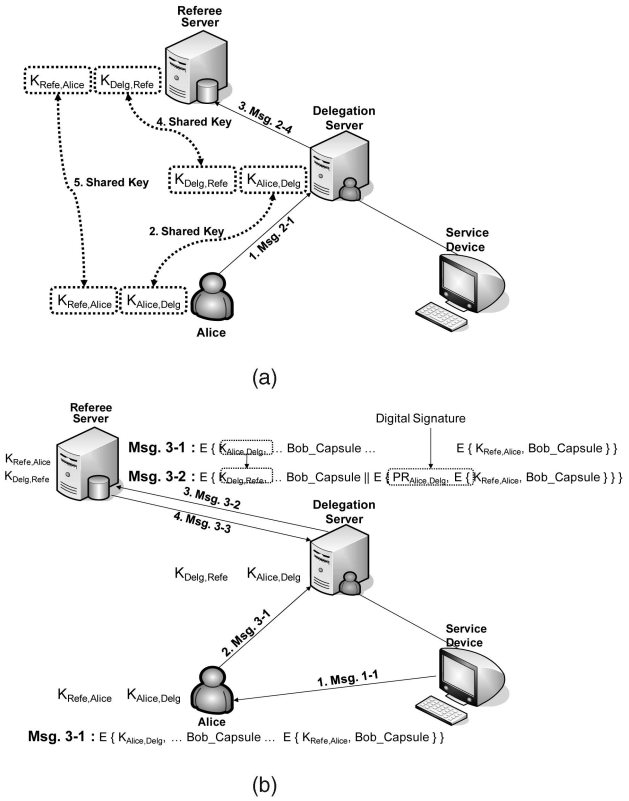


Fig. 15. Overall message transactions for the nonrepudiation of PKASSO. (a) Registration phase for nonrepudiation (State 2). (b) Verification phase for nonrepudiation (State 3).

4.3.1 Registration Phase for Nonrepudiation (State 2)

As an initial phase of nonrepudiation, the delegation server registers the delegated user information ID_{Alice} and $K_{Refe,Alice}$ with the referee server, as described in Fig. 15a:

1. A delegation request message, i.e., Message 2-1, is used for sharing two keys exclusively with the delegation server and the referee server, that is, it provides nonrepudiation for a state in which a user's authentication is delegated. To share keys securely and exclusively, the shared keys are encrypted and signed with the user's private and public keys (PU_{Delg}, PU_{Refe}).
2. In reply to the message, the delegation server stores the key that is to be shared with *Alice* by the decryption of Message 2-1 and generates a proxy certificate over Messages 2-1 and 2-2. Next, the delegation server injects a symmetric key that is to be shared with the referee server and generates Message 2-4 by using a private key PR_{delg} and a public key PU_{refe}.
3. In response to Message 2-4, the referee server checks the validity of the message and stores two different symmetric keys that are to be shared with the delegation server and *Alice*, respectively. As a result, through this registration phase, the three entities, namely, user, delegation server, and referee server, exclusively share three keys ($K_{Delg,Refe}$, $K_{Refe,Alice}$, and $K_{Alice,Delg}$).

4.3.2 Verification Phase for Nonrepudiation (State 3)

Once registration is successfully completed in *State 2*, the referee server checks the validity of the user's message for each authentication transaction, as described in Fig. 15b. To support nonrepudiation, the referee server generates binding information between a device and authentication messages as follows:

1. *Alice* receives a challenge message from a service device, i.e., Message 1-1. The challenge message, called the $Bob\_Capsule$, is the output of SHA-1 with two inputs: a serial number of the service and a randomly generated nonce value. Due to the NP-completeness of SHA-1, only the service device can perceive the nonce value.
2. *Alice* sends the delegation server an authentication request message, i.e., Message 3-1, which contains two $Bob_Capsule$s. One of the $Bob\_Capsules$ is encrypted with $K_{Refe,Alice}$ and the other is encrypted using $K_{Alice,Delg}$.
3. In reply to Message 3-1, the delegation server sends a $Bob\_Capsule$ verification message, i.e., Message 3-2, to verify *Alice*'s message against malicious user behavior. Message 3-2 contains a digital signature of the encrypted $Bob\_Capsule$ using the delegated private key $PR_{Alice,Delg}$ and the $Bob\_Capsule$. These contexts are encrypted with $K_{Delg,Refe}$.
4. The referee server compares the two $Bob\_Capsules$ to check the justness of the authentication request message. If the two messages are identical to each other, the referee server sends an OK message to the delegation server. Otherwise, the referee server sends a BAD message to the delegation server. Subsequently, the verified $Bob\_Capsule$ and the digital signature of the delegation server are retained on the local repository of the referee server.

4.3.3 How the User Authentication Records Can Be Proved

As shown in Fig. 16, the referee server retains the history data, which states that *Alice* sent Message 3-1 as a type of a notarized authentication list (NAL). The NAL is the data structure for storing the evidence of nonrepudiation, and all of the contexts are periodically stored with the signature of the referee server. This process, which ensures the integrity of user authentication by using the digital signature of the referee server, contains the following data:

- Data stored per delegation:
  - ID_{Alice}: Message 2-4.
  - $K_{Refe,Alice}$, $K_{Delg,Refe}$: Message 2-4.
  - Seq_{Alice}: Message 2-5.
- Data stored per authentication:
  - $Bob\_Capsule$: Message 3-2.
  - $E\{K_{Refe,Alice}, Bob\_Capsule\}$: evidentiary data.

Fig. 17 illustrates how the NAL is used to prove that *Alice* sent an authentication request message to the delegation server. Let us assume that a service device asserts that *Alice* repudiates an authentication of the service device. In this

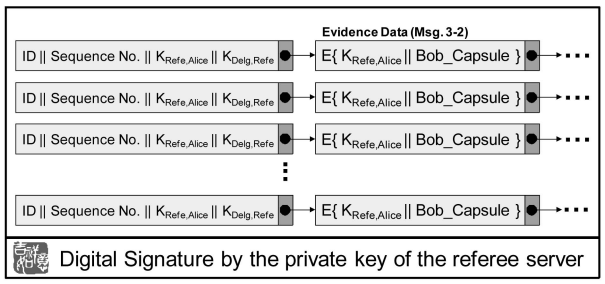


Fig. 16. Notarized Authentical List (NAL) data structure.

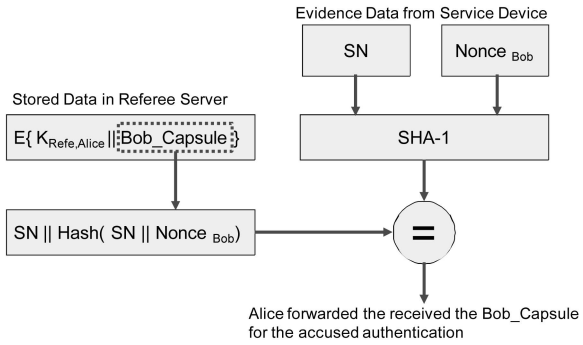


Fig. 17. Nonrepudiation mechanism using history data (NAL) in PKASSO.

case, the service device can put in a claim for a justice with the serial number included in the challenge message $Bob\_Capsule$ to the referee server. The referee server then requires the evidentiary data that were used to generate the challenge message with the serial number and inputs the evidence data and the serial number to SHA-1. If the output of SHA-1 is identical to the stored data in the referee server, it proves that *Alice* forwarded the received $Bob\_Capsule$ for the accused authentication. The referee server can therefore refute *Alice*'s repudiation.

4.4 Security Considerations of PKASSO

We analyzed the protocol safety by considering the replay attacks and man-in-the-middle (MITM) attacks. We assumed that the underlying cryptography AES, RSA, and SHA were invulnerable with regard to message secrecy and integrity; hence, we did not consider attacks such as cryptanalysis and message slicing. On the other hand, any principle can place or inject a message on any link at any time. In addition, any principle can eavesdrop, drop, alter, or redirect all exchanged messages being passed along any link or replay messages recorded from past communications. In this section, we confirm the safety of our proposed protocol in relation to replay and MITM attacks.

Theorem 1. *PKASSO is safe from replay attacks.*

Proof. Let us assume that the authentication path is $[Alice \Leftrightarrow \text{Delegation Server} \Leftrightarrow \text{Kerberos Server} \Leftrightarrow Bob]$. We prove the safety for the next attack types as follows:

1. A replay attack for the delegation request message, i.e., Message 2-1. In this case, an intruder can try to attack by sending the captured delegation request message. However, the key to be shared with the delegation server cannot be read by the intruder because the key is encrypted with the public key of the delegation server. Thus, the intruder can no longer perform the delegation mechanism.
2. A replay attack for the delegation response message, i.e., Message 2-2. In this case, an intruder can try to attack by sending the captured delegation response message. However, the intruder's attack cannot succeed because the delegation request message includes the nonce data enclosed in the delegation request message.
3. A replay attack for the authentication request message, i.e., Message 3-1, and
4. a replay attack for the response message between the delegation server and the service device, i.e., Message 6-2. In these cases, an intruder cannot reuse the authentication request message, i.e., Message 3-1, or the response message, i.e., Message 6-1, because the $Bob\_Capsule$ included in the challenge message is altered in each authentication. □

Theorem 2. *PKASSO is safe from MITM attacks.*

Proof. We prove the safety for the next attack types as follows:

1. An MITM attack between a user and a delegation server. In this case, each entity performs a mutual authentication over PKI before a delegation and shares the key for a secure connection. Thus, the intruder cannot forge the authentication request message of the user.
2. An MITM attack between a user and a service device and
3. an MITM attack between a delegation server and a service device. In these cases, each service device generates a $Bob\_Capsule$, which is altered for each authentication, and this capsule is encrypted and transmitted by using a shared key that is generated in a previous state. Thus, the intruder cannot successfully masquerade as the user or the service device. □

5 Efficiency and Performance Evaluation

In this section, we present the performance results obtained with our prototype implementation of PKASSO. First, we demonstrate the overall experimental environment. We then describe the cryptography operation experiment and the operational efficiency of the authentication protocol to evaluate the performance of PKASSO in terms of authentication latency.

5.1 Experimental Environment

To evaluate the performance characteristics of PKASSO, we constructed a pseudoservice device and a PANDA which are coupled to a load generator. The objective of the pseudodevices is to simulate the processing and communication resources anticipated in a full implementation (for multiusers and multiservice devices). Fig. 18 shows the

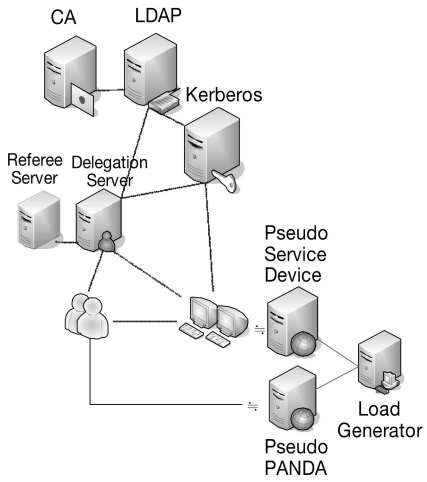


Fig. 18. Experiment environment to measure the performance of PKASSO in terms of authentication latency and operation efficiency in PANDA.

overall experimental environment. The pseudoservice device and PANDA have operation times that are similar to actual operation times (for example, those that result from communications, encryption and decryption, and message processing). The pseudodevices are modeled on the cryptography operation (RSA and AES) times, the data rate, the detection ratio, and the delivery latency of ZigBee communications. Moreover, these pseudodevices are connected to PKASSO and receive control signals from the load generator. The load generator is a module that generates control signals to produce authentication request messages by using a random generator that models the mobility of users [16].

5.2 Cryptography Operation Experiment

The first experiment measures the cryptography processing time on ZigBee devices (PANDA and a service device) equipped with an ATmel 8-bit processor (16 MHz) [8] and a server (a 3.2 GHz Xeon processor with a 4 Gbyte RAM). Fig. 19 compares the processing time of an RSA 1,024 bit algorithm as an asymmetric key operation, an AES 128 bit algorithm as a symmetric key operation, and a SHA-1 algorithm as a hash function. It shows that the times required to decrypt and to encrypt a 128 byte block of data with the RSA 1,024 bit algorithm are 4,723 and 226 ms, respectively, on ZigBee devices. On the other hand, the times required to encrypt and to decrypt the block with the AES 128 bit algorithm on the device are both 3 ms because an AES accelerator is embedded in the ZigBee communication module (CC2420 [17]). Furthermore, SHA-1 needed 6 ms of operation time to generate a challenge message. On the server side, the cryptography operation time is drastically reduced by a high-performance processor and a huge memory.

5.3 Authentication Protocol Efficiency

The performance of the authentication protocol in terms of authentication latency and the consumption of processing and communication resources is an important factor to be considered when designing authentication protocols [18]. In this section, we analyze the efficiency of five different

Platform	Cryptography		Operation Time
• PANDA • Service Device - CPU : ATmega1280 - RAM : SRAM 256KB	RSA 1024bit	Private Key	Avg. 4723ms
		Public Key	Avg. 226ms
	AES 128bit	Encryption	Avg. 3ms
		Decryption	Avg. 3ms
	Hash Function	SHA-1	Avg. 6ms
• Server - CPU : Xeon 3.2GHz - RAM: 4GB	RSA 1024bit	Private Key	Avg. 2.917ms
		Public Key	Avg. 0.170ms
	AES 128bit	Encryption	Avg. 0.006ms
		Decryption	Avg. 0.006ms

Fig. 19. Processing times of encryption/decryption for each algorithm and operation environment.

authentication protocols (PKIX, Kerberos, M-PKINIT, NSI, and PKASSO) in terms of computation and communication efficiency.

5.3.1 Computation Efficiency

Fig. 20 gives the number of public and private keys and the symmetric key operations performed with the total operation time for each authentication protocol. Despite having the smallest number of cryptography operations per authentication, the PKIX protocol has the longest operation time among these protocols because it has the largest number of private key operations in the resource-constrained mobile device. Kerberos, on the other hand, has the shortest operation time because the Kerberos authentication can be completed by symmetric key operations. In the case of PKASSO, the first time that a user accesses the security infrastructure, the user delegates the user authentication operations to the delegation server, which requires two public key operations and one private key operation. The delegation operation time of PKASSO is similar to the operation time of an M-PKINIT TGT or a PKIX. After the delegation operation, the user can initiate self authentication by processing only five symmetric key operations; this process has a much shorter operation time than the corresponding process of an M-PKINIT TGT and a PKIX.

By way of summarizing the above results, we give an outline of the minimum operation time of the authentication protocols. The PKIX operation time with the RSA 1,024 bit algorithm is 4,952.34 ms. The Kerberos has a much

System	Mobile			Server			Total operation time
	Pu	Pr	S	Pu	Pr	S	
PKIX(RSA-1024bit)	1	1	1	2	0	0	4952.34 ms
Kerberos	0	0	8	0	0	6	24.04 ms
M-PKINIT TGT	1	1	7	1	1	5	4973.12 ms
M-PKINIT SGT	0	0	8	0	0	4	24.02 ms
NSI	0	1	1	3	0	0	4726.51ms
PKASSO Delegation	2	1	2	4	5	2	5196.28 ms
PKASSO TGT	0	0	5	1	1	12	18.16 ms
PKASSO SGT	0	0	5	1	1	7	18.13 ms

Pu: The number of public key operation for each authentication
Pr: The number of private key operation for each authentication
S: The number of symmetric key operation for each authentication

Fig. 20. The number of public/private keys and the symmetric key operations with the total operation time for each protocol (PKIX, Kerberos, M-PKINIT, NSI, and PKASSO).

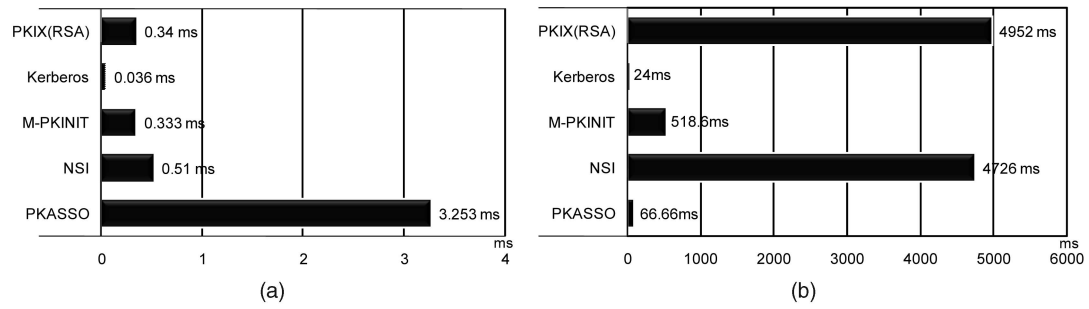


Fig. 21. Operation time (a) on the server side and (b) on the client side for each authentication protocol.

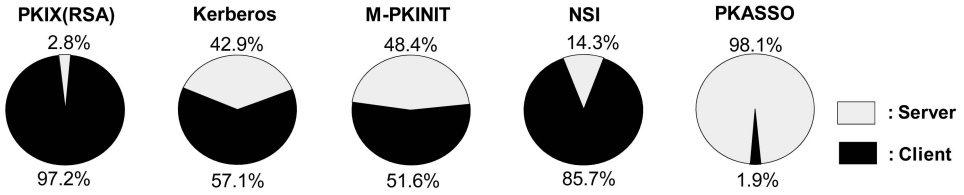


Fig. 22. Distribution of client side operations and server side operations.

shorter operation time (24.04 ms) than the PKIX, which is due to the fact that the symmetric key operation has a much shorter operation time than the asymmetric key operation. In the case of the M-PKINIT protocol, when a user acquires a TGT, the operation time is similar to the PKIX operation time. On the other hand, the operation time for obtaining an SGT with the M-PKINIT protocol is 24.02 ms because the user can grasp an SGT by using the TGT without asymmetric key operations. The NSI has shorter operation time (4,726.51 ms) than the PKIX, which is due to the PKI server that is responsible for searching and verifying certificates on behalf of a mobile device. In the case of PKASSO, even though the delegation operation of PKASSO has the longest operation time among all the cryptography operations, an authentication can be accomplished without asymmetric key operations after the delegation operation. As a result, we confirmed that PKASSO has the shortest operation time after the delegation of the user's authentication, even though PKASSO provides a PKI-based authentication. This outcome is due to the offloading of the asymmetric key operation from the user's device to the delegation server in *State 2*.

In order to measure how many cryptography operations on the client side cause an obstructive authentication latency, we estimated the operation time of the server side and the client side for each authentication protocol, as shown in Fig. 21. In the case of the Kerberos and M-PKINIT protocols, we assumed that 10 percent of all of the authentications were for acquiring a TGT, while the remainder was for obtaining an SGT. In the case of PKASSO, 10 percent of the TGT requests were delegation requests [16]. Fig. 21a presents a comparison of the operation time of the server side. Fig. 21b compares the operation time performed on the client side. Even though the operation time of PKASSO on the server side is 10 times longer than that of PKIX, the operation time of PKASSO on the client side is much shorter than that of PKIX. As a result, the total operation time of PKASSO (69.91 ms) is much shorter than PKIX (5,178.34 ms). It is due to the gap in computation power between the server, the resource-restricted mobile devices, and the distribution of client-side operations and server-side operations, as shown in Fig. 22. The ratio of the client-side operations to the server-side operations is 0.972 to 0.028 for the PKIX protocol, 0.571 to 0.429 for the Kerberos protocol, 0.516 to 0.484 for the M-PKINIT protocol, 0.857 to 0.143 for the NSI, and 0.019 to 0.981 (the smallest ratio) for the PKASSO protocol.

Despite having the longest operation time on the server side of PKASSO, the overall operation time of PKASSO (69.91 ms) is 74.07 times shorter than that of PKIX (5,178.34 ms). The reason for this result is that only 1.9 percent of the authentication operations of PKASSO are executed in a resource-constrained mobile device, whereas 97.2 percent of PKIX operations are executed in such a device.

5.3.2 Communication Efficiency

The comparison of the required payload per authentication is illustrated in Fig. 23. In the case of Kerberos, M-PKINIT, and PKASSO, we assumed that 10 percent of all of the authentications were initiated to acquire a TGT and that the remainder were for obtaining an SGT. In terms of the communication overhead, PKASSO is the most efficient protocol because its smallest payload (0.076 Kbyte) enables the communication overhead of PKASSO to be minimized.

The overall data payload of PKIX and NSI is about 3.67 Kbyte and 1.76 Kbyte due to the transmission of each entity's certificate. In the case of M-PKINIT, the TGT request has the heaviest data payload (7.54 Kbytes) because the user should send and receive each entity's certificates and a TGT for each movement to other Kerberos realms. Furthermore, 2.35 Kbyte transmissions are required to acquire an SGT in M-PKINIT and Kerberos. In the case of PKASSO, whenever the user first accesses the security infrastructure, a 1.42 Kbyte transmission over ZigBee is required to delegate the user's authentication to the delegation server. After the delegation, the PKASSO authentication has the smallest payload (0.332 Kbyte) for obtaining a TGT or an SGT because the user authentication is completed by the transmission of a $Bob\_Capsule$ (0.032 Kbyte) and a confirmation message (0.044 Kbyte).

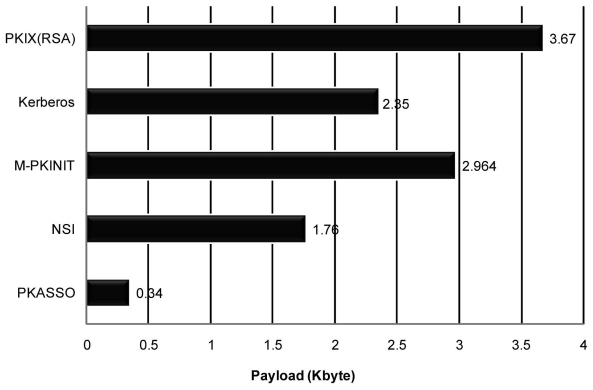


Fig. 23. Required payload on each authentication protocol.

5.4 Performance Evaluation

Fig. 24 illustrates the improvement in the authentication latency with our scheme and compares it with a general PKIX operation equipped with a PANDA and a smart card [19]. In the case of the Kerberos and M-PKINIT protocols, we assumed that 10 percent of all of the authentications were for acquiring a TGT, while the remainder was for obtaining an SGT. In the case of PKASSO, 10 percent of the TGT requests were delegation requests [16]. If an RSA 1,024 bit algorithm is processed on a PANDA equipped with an 8 bit processor (16 MHz) [8], the authentication latency averages to 5.01 seconds. Furthermore, if an authentication with Kerberos, M-PKINIT, and NSI is executed on the above platform, the authentication latency averages 0.19, 0.74, and 4.75 seconds, respectively, because, in order to obtain a TGT, the Kerberos authentication protocol uses symmetric key operations and M-PKINIT uses asymmetric key operations. With the M-PKINIT protocol, a user can get an SGT without asymmetric key operations; hence, the authentication latency of M-PKINIT is much shorter than that of PKIX. The authentication latency of NSI is shorter than PKIX, which is due to the delegated operations for the certificate verification by the PKI server. The latency of a contact-type smart card is estimated to be 3.70 seconds [20], which is faster than the latency of PKIX on our security device. In the case of PKASSO, even though the delegation operation of PKASSO takes longer than a general PKIX authentication (5.19 seconds), the authentication latency of PKASSO with a PANDA can be shortened to

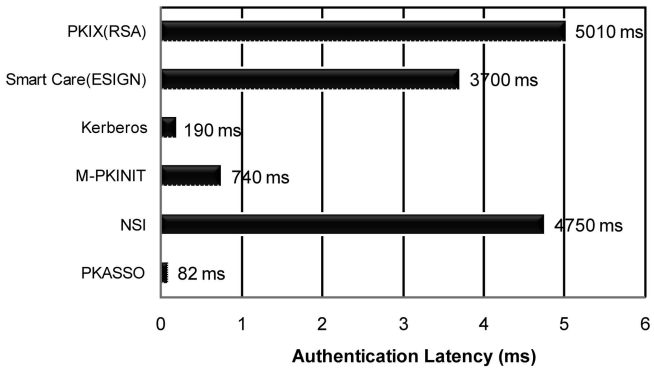


Fig. 24. Authentication latency for PKIX (RSA), Smart Card, Kerberos, M-PKINIT, NSI, and PKASSO.

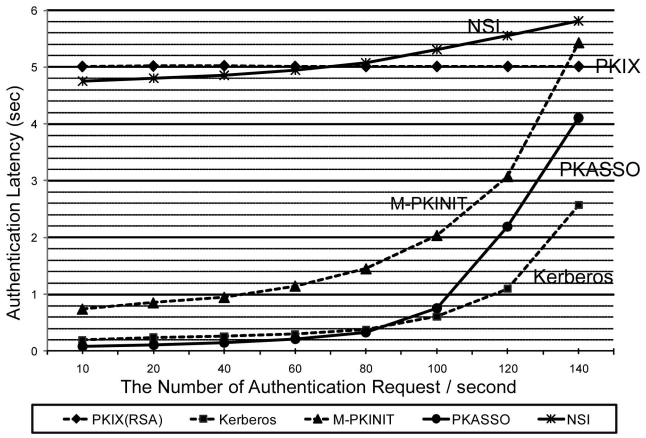


Fig. 25. Authentication latency with varying numbers of authentication requests per second for PKIX (RSA), Kerberos, M-PKINIT, NSI, and PKASSO.

0.082 second for a specified period after the delegation. As described in Section 5.3, the major reduction in the authentication latency is due to the offloading of complex operations from the devices to the infrastructure. As a result, we can minimize the authentication latency from an average of 5.01 to 0.082 second without compromising the security level of PKIX.

Fig. 25 shows how the authentication latency mutates as the number of authentication requests per second varies. The number of authentication requests per second ranges from 10 to 140, with 87,880 users and 17,576 service devices. For the M-PKINIT, Kerberos, NSI, and PKASSO protocols, we found that the authentication latency increases as the number of authentication requests increases, mainly as a result of the cryptography operations and the communication overhead on the server side. In the case of M-PKINIT, the authentication latency increased from 0.733 to 5.424 seconds as the number of authentication requests per second increased. The authentication latency of NSI is elevated from 4.750 to 5.812 seconds with varying numbers of authentication requests per second. In the case of PKIX (RSA), there was almost no variation in the authentication latency because the server-side overhead of PKIX was much smaller than the client-side overhead. In the case of PKASSO, the authentication latency is shorter than that of Kerberos in the state where the number of requests is less than 90. However, the latency of PKASSO is longer than that of Kerberos and has a higher rate of increase if PKASSO receives more requests than 90. This phenomenon is due to the fact that the quantity of server-side operations of PKASSO is much greater than that of Kerberos. This result confirms that our PKI-based authentication with nonrepudiation can be accomplished seamlessly by PKASSO whenever the number of requests per second is less than 90.

6 Conclusion

Our task has been to provide a full-fledged security solution tailored for a ubiquitous and mobile computing environment, where numerous devices and sensors with severe resource-constraints interact with each other. To accomplish this task, we thoroughly reviewed the ways by which the conventional PKI-based security infrastructure is used in the environment and we consequently derived the blueprints for an efficient PKI-based security infrastructure

called PKASSO. Besides utilizing conventional PKI entities in our security infrastructure, we conceived and implemented the concepts of a delegation server, a referee server, and a new PKI-based SSO protocol. Our security infrastructure features three remarkable achievements:

1. **PKI-based SSO protocol.** This protocol enables an operationally efficient security mechanism and cost-effective deployment of the security services by offloading complex PKI operations from the mobile devices to the infrastructure.
2. **Delegation server.** This server is responsible for performing prohibitively expensive PKI operations on behalf of a PANDA without compromising the security level of the PKI. As a result, it minimizes the computational overhead of the security device.
3. **Referee server.** This server ensures nonrepudiation of any transaction between a delegator and a delegatee.

Our infrastructure consequently enables a cost-effective but uncompromisingly secure development of diminutive security devices. Furthermore, our delegation mechanism significantly improves the authentication latency. According to the performance evaluation, the authentication latency of our infrastructure (which averages 0.082 second) is much shorter than the authentication latency of a contact-type smart card (which averages 3.70 seconds) or a conventional PKI-based authentication latency (which averages 5.01 seconds).

REFERENCES

[1] J.I. Hong, J.D. Ng, S. Lederer, and J.A. Landay, "Privacy Risk Models for Designing Privacy-Sensitive Ubiquitous Computing Systems," *Proc. Fifth Conf. Designing Interactive Systems,* pp. 91-100, 2004.

[2] J. Lee, S.-H. Lim, J.-W. Yoo, K.-W. Park, H.-J. Choi, and K.H. Park, "A Ubiquitous Fashionable Computer with an i-Throw Device on a Location-Based Service Environment," *Proc. 21st IEEE Int'l Conf. Advanced Information Networking and Applications Workshops,* vol. 2, pp. 59-65, 2007.

[3] H. Seok, K.-W. Park, S.S. Lim, and K.H. Park, "Implementation of U-Kiosk Based on Panda and VNC," *KISS,* vol. 33, no. 2A, pp. 238-243, http://uci.or.kr/G300-c15985164.v33n2Ap238, 2007.

[4] D. Cotroneo, A. Graziano, and S. Russo, "Security Requirements in Service Oriented Architectures for Ubiquitous Computing," *Proc. Second Workshop Middleware for Pervasive and Ad Hoc Computing,* pp. 172-177, 2004.

[5] M. Fahrmair, W. Sitou, and B. Spanfelner, "Security and Privacy Rights Management for Mobile and Ubiquitous Computing," *Proc. Seventh Int'l Conf. Ubiquitous Computing,* 2005.

[6] K.-W. Park, H. Seok, and K.-H. Park, "PKASSO: Towards Seamless Authentication Providing Non-Repudiation on Resource-Constrained Devices," *Proc. 21st IEEE Int'l Conf. Advanced Information Networking and Applications Workshops,* vol. 2, pp. 105-112, 2007.

[7] P Working Group, http://www.ietf.org/html.charters/pkix-charter.html, 2008.

[8] *ATmega1280 Datasheet: 8-bit Microcontroller with 256 Kbytes In-System Programmable Flash.* ATMEL Press, 2007.

[9] A.A. Pirzada and C. McDonald, "Kerberos-Assisted Authentication in Mobile Ad Hoc Networks," *Proc. 27th Australasian Conf. Computer Science,* pp. 41-46, 2004.

[10] L. Zhu and B. Tung, *RFC 4556: Public Key Cryptography for Initial Authentication in Kerberos (PKINIT).* IETF Network Working Group, 2006.

[11] A. Harbitter and D.A. Menascé, "The Performance of Public Key-Enabled Kerberos Authentication in Mobile Computing Applications," *Proc. Eighth ACM Conf. Computer and Comm. Security,* pp. 78-85, 2001.

[12] M. Jalali-Sohi and P. Ebinger, "Towards Efficient PKIS for Restricted Mobile Devices," *Proc. IASTED Int'l Conf. Comm. and Computer Networks,* pp. 42-47, 2002.

[13] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, *RFC 3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile.* IETF Network Working Group, 2004.

[14] K.-W. Park, S.S. Lim, and K.H. Park, "Ultra-Low-Power Security Card, PANDA, for PKI-Based Authentication and Ubiquitous Services," *Proc. Conf. Next-Generation Computing,* pp. 367-373, 2006.

[15] *ZigBee Specification v1.0.* ZigBee Alliance Board of Directors, 2005.

[16] A. Gellert and L. Vintan, "Person Movement Prediction Using Hidden Markov Models," *Studies in Informatics and Control,* vol. 15. ISI Thomson INSPEC, 2006.

[17] *CC2420 Datasheet 2.4 GHz IEEE 802.15.4/ZigBee-Ready RF Transceiver.* Chipcon Press, 2006.

[18] A. Harbitter and D.A. Menascé, "A Methodology for Analyzing the Performance of Authentication Protocols," *ACM Trans. Information and System Security,* vol. 5, no. 4, pp. 458-491, 2002.

[19] Hitachi, *Single-Chip Microcomputer H8/2168 Group Hardware v3.0.* Renesas Technology, 2004.

[20] C.-H. Yang, H. Morita, and T. Okamoto, "Security and Performance Evaluation of ESIGN and RSA on IC Cards by Using Byte-Unit Modular Algorithms," *IEICE Trans. Comm.,* vol. E88-B, no. 3, pp. 1244-1248, http://ietcom.oxfordjournals.org/cgi/content/abstract/E88-B/3/1244, 2005.

Ki-Woong Park received the BS degree in computer science from Yonsei University in 2005 and the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007. He is currently working toward the PhD degree in the Division of Electrical Engineering at KAIST. His research interests include security protocol, network security, and embedded systems. He is a student member of the IEEE and the IEEE Computer Society.

Sang Seok Lim received the BS degree in computer engineering from Chonnam National University in 1998 and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2000 and 2006, respectively. He is currently a senior software engineer at Samsung Electronics, Suwon, Korea. His research interests include embedded systems and their applications, high-performance server systems, and operating systems. He is a member of the IEEE.

Kyu Ho Park received the BS degree in electronics engineering from Seoul National University in 1973, the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology(KAIST) in 1975, and the DrIng degree in electrical engineering from the University of Paris XI in 1983. Since 1983, he has been a professor in the Division of Electrical Engineering and Computer Science at KAIST. From 2005 to 2006, he was the president of the Korea Institute of Next-Generation Computing. His research interests include computer architectures, file systems, storage systems, ubiquitous computing, and parallel processing. He is a member of KISS, KITE, the Korea Institute of Next-Generation Computing, the IEEE, and the ACM.