Debug Port Protection Mechanism for Secure Embedded Devices

Keun-Young Park, Sang-Guun Yoo, and Juho Kim

Abstract—**In this paper we propose a protection mechanism for the debug port. While debug ports are useful tools for embedded device development and maintenance, they can also become potential attack tools for device hacking in case their usage is permitted to hackers with malicious intentions. The proposed approach prevents illicit use of debug ports by controlling access through user authentication, where the device generates and issues authentication token only to the server-authenticated users. An authentication token includes user access information which represents the user's permitted level of access and the maximum number of authentications allowed using the token. The device authenticates the user with the token and grants limited access based on the user's access level. The proposed approach improves the degree of overall security by removing the need to expose the device's secret key. Availability is also enhanced by not requiring server connection after the initial token generation and further by supporting flexible token transfer among predefined device groups. Low implementation cost is another benefit of the proposed approach, enabling it to be adopted to a wide range of environments in demand of debug port protection.**

Index Terms—**Debug port, device hacking, authentication token**

Manuscript received Jul. 11, 2011; revised Jan 14, 2012
Dept. of Computer Science & Engineering, Sogang University, Korea
E-mail : jhkim@sogang.ac.kr (Corresponding Author)

I. INTRODUCTION

Enhanced performance and features of modern embedded devices have brought on a variety of services utilizing those resources for business, which at the same time have invited device hackings aimed at feature modification, illegal use of paid services, or leaking sensitive information. Fast-evolving device hacking techniques necessitate device manufacturers to design a hack-proof, safe, and reliable execution environment built up from the hardware platform level with security in mind from early on [1].

Device hacking through debug ports or service ports is a well-recognized security threat which must be addressed in the device platform design. Most embedded devices have debug ports for utility purposes such as testing, programming or maintenance, and those ports can be used for obtaining low-level device-internal information which can give device hackers considerable edge unless securely controlled [2, 3]. For instance, an attacker using a debugging tool compliant with IEEE 1149.1 standard test access port can easily access the processor or the memory-internal information of a device equipped with the port [4, 5]. With access to low-level internal information, the attacker can opt to mount an attack much more penetrative than a purely software-based attack. In case a software is executing computations to verify a user or a license copy, the attacker can analyze the key values used in the computations from information collected from the registers or the memory [6, 7]. In other cases, the attacker can download and modify a device's firmware to prevent or bypass the execution of a security routine, or analyze security vulnerability and develop a malware to exploit it [2].

Therefore, to ensure protection against device hacking, debug port access must be controlled.

In this paper, we propose a protection mechanism which provides user authentication and access control capabilities for the purpose of preventing unauthorized access by hackers at the cost of minimal availability degradation. After the introduction in Section I, we provide an overview of existing debug port protection mechanisms and their limitations in Section II. In Section III and IV, we elaborate the details of the proposed approach, including its procedures, policies, and architecture. Section V analyzes the security of the proposed approach, while Section VI provides a comparison of the proposed approach against related works. In Section VII we illustrate the implementation results, and lastly in Section VIII, we conclude the paper with a summary.

II. RELATED WORKS

Security and availability are typically found to be in a trade-off relationship, where previous works on debug port security mechanisms have failed to find a satisfactory common ground and suggested emphasis on only one of the traits while deemphasizing the other.

Given the purpose of debug port protection is to prevent hackers from gaining access to unauthorized information, blowing fuses to physically remove debug ports from devices on which further debugging is deemed unnecessary can achieve that security goal [8]. However, considering the fact that debug ports serve as a useful maintenance tool, its constant availability must be ensured on most devices. That is, authorized users must be able to use a debug port whenever necessary.

User authentication needs to be applied to debug ports to prevent illegal exploitation by attackers while ensuring accessibility for normal users. Previous works have proposed two separate approaches to user authentication, which are two-entity authentication and three-entity authentication. The two-entity authentication uses a shared secret key between the device and the user [9, 10]. A user with the key can authenticate oneself regardless of time or place, minimizing the loss of availability while maintaining a certain level of security. However, the key is exposed to the external world while being directly delivered to and managed by the user, limiting the security level achievable by the approach. The three-entity authentication, on the other hand, utilizes a separate server to authenticate a user for improved security [11]. In this approach, the server authenticates the user and delivers the authentication result along with the user's authorization information to the device through a asymmetric key encryption channel established between the server and the device. This approach offers higher level of security by hiding the key from the user, and supports various security policies for user authentication and authorization on the server. However, the need for the server to authenticate the user on every debug port use requires continuous communication with the server, disabling debug port use and lowering overall availability when network is unavailable.

Complementing the previous approaches' shortcomings while combining their strengths, [12] has proposed a user authentication method in which the server issues to the authenticated user a credential with which to authenticate oneself to a particular device. The device verifies user-submitted credentials and opens the debug port to only the users with valid credentials. In contrast to the other two authentication schemes, the approach taken by [12] effectively eliminates the concern of key management and removes the need for networking with the server after a credential has been issued. However the approach still faces several limitations regarding security and availability. First, the user must issue from the server separate credentials for each device to be tested, and the user can't use the debug port of a device if the particular device is located in an area without available networking because no credential can be issued. Secondly, the approach does not provide any built-in mechanism to restrict the use of a credential already issued by a server. Additional security mechanism for tracking the accumulative usage count of each credential is necessary to provide such function. Without additional control measures to revoke a previously-issued credential, the user has unlimited usage of the credential. Lastly, the user cannot change the password of an issued credential. If a credential's password is leaked while in use, the credential has to be discarded and re-issued from the server. This places a severe restriction on credential usage in an environment where multiple users test a single device. For instance, if a credential has to be shared even briefly with another user, the credential owner must discard and re-issue the credential since its

password has been exposed.

In this paper, we propose a debug port protection mechanism which resolves the limitations of previous works and provides improved security and availability.

III. PROPOSED APPROACH

In this paper, we propose a new debug port protection mechanism with improved security and availability. The proposed approach effectively prevents device hacking attempts on the debug port by using authentication tokens to authenticate users and by controlling access to the port. In this Chapter we explain the details of the proposed approach's user authentication and access control procedures.

1. User Authentication Scheme using Authentication Token

The proposed approach establishes a debug port protection mechanism between the device, the server and the host, as shown in Fig. 1, where the role of the host is to relay the communication between the device and the server while providing user interface for the user. In our approach, the device authenticates the debug port user using the authentication token and secret for the token submitted by the user and determines the user's access level based on the access level authorized for the user's token. The server authenticates the user and authorizes the user with appropriate access level, which enables the device to issue a corresponding authentication token for the user. The token includes access level information of the owner and the maximum number of authentications allowed (MNAA) using the token. Therefore a user who wishes to use the debug port of a device must first have the device issue an authentication token for the user. Once a token has been issued, no further communication with the server is necessary while there are remaining authentications allowed by the token.

The authentication token generation procedure is a challenge-response interaction between the device, the host and the server, which proceeds in the following steps: First, the user inputs the necessary user account information and secret to be used for token generation at the device. Then the host sends an authentication requests to the device and obtains a device-generated challenge.

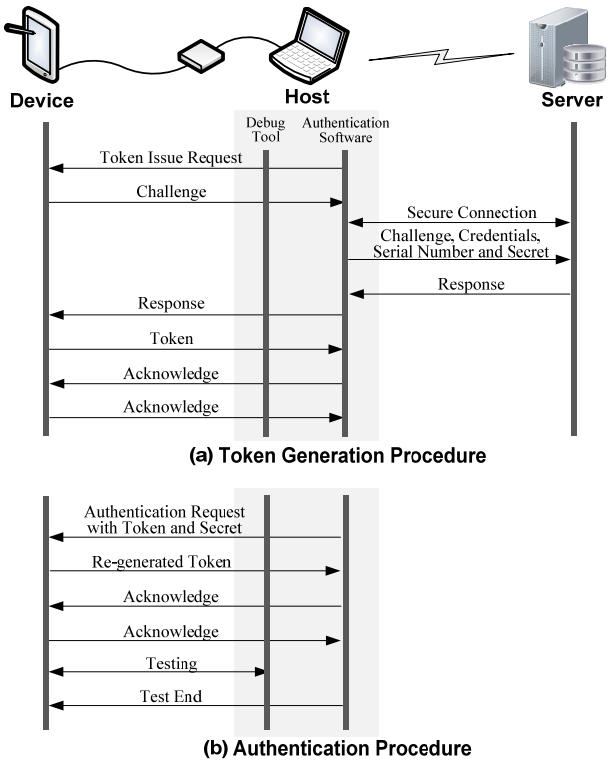


Fig. 1. User Authentication Scheme.

Thereafter the host establishes a secure communication channel with the server and requests for user authentication and authorization by sending to the server the user's account information and secret, the device's serial number, and the device-generated challenge. The server, upon receiving the necessary information, authenticates the user and checks the user's device access level. A response confirming that the user has been authenticated is generated by the server, which includes the user's access level information, the secret, and the MNAA value of the token to be generated at the device. The server response is delivered to the device, in which the authentication module verifies the response in order to check if the user has been properly authenticated by the server. If so, the module generates a token corresponding to the user's access level and MNAA. Afterwards, the token generation information (TI) is stored within the non-volatile storage of the device for the purpose of future token verification, and the token itself is delivered to the host, completing the authentication token issue procedure.

The user authentication procedure is the sequence of steps to authenticate a token-holder in order to permit access to the debug port under correct restrictions

imposed by the user's access level and MNAA value. The procedure is carried out on a one-on-one connection between the device and the host, during which the device verifies the user-provided token and secret to authenticate the user and to control the user's access to the debug port. The first step of the procedure is initiated by the user requesting the host for authentication. The next step is followed by the host sending the user-provided token and its secret to the device. The device, upon receiving the user's token and secret, verifies them with the corresponding TI previously stored within its non-volatile storage in order to check that the token is valid and that the user is indeed the correct owner of the token. If the authentication is successful, the device subtracts 1 from the token's MNAA value to obtain MNAA' and reflects this changed value to the existing token by re-generating the token with MNAA' and storing the resulting TI at the TI storage. After the re-generated token is sent back to the host, the device grants access to the user based on the user's access level.

2. Discarding Expired Authentication Token

A token becomes expired during user authentication when its MNAA value reaches zero after subtraction, or when a replacement token is generated at the device using the subtracted MNAA value (MNAA'). An expired token must be discarded to prevent further use. When a replacement token is generated during user authentication, the token's corresponding TI is also updated with the new MNAA'. If a token with MNAA value of 1 has used up its final authentication, TI for the token is deleted from the non-volatile storage to disable further use of the token before access to the debug port is granted. Even valid tokens can be discarded at the discrimination of the user or the device. For instance, the user can request for the device to discard an authentication token if it serves no more purpose for the user or if the secret for the token has been leaked. Further, the device can initiate a token discard if the number of consecutive authentication failures using a token reaches a certain threshold, which thereafter will be perceived by the device as a brute force attack.

3. Group Token Issue & Token Transfer Scheme

The proposed approach can, depending on the manufacturer's security policy or key allocation method, support the use of device group authentication tokens which permit authentication of multiple devices belonging to a certain group of devices. A device manufacturer can, for instance, assign an identical key to a selected device group defined by such criteria as device model name, hardware version, or feature sets. This practice of device group key management can help to reduce chip fabrication costs or to increase operation efficiency. When device group authentication token is in use, a user can transfer his authentication token issued by a device to another device to authentication himself given that both the source device and the destination device belongs to the same device group and the MNAA value of the transferred token is greater than zero.

The user wishing to make a token transfer must first be authorized to own a group authentication token and have an authentication token issued from a device which belongs to the same group as the destination device. The authentication token issue procedure is shared between group and non-group tokens, with the only difference in that the server permits group authentication token ownership only to authorized users, while the device generates group authentication tokens only for users with confirmed ownership.

The user owning a group authentication token can transfer the token to other devices within the same device group if desired. The authentication token transfer procedure, as shown in Fig. 2, employs a challenge-response interaction between the source device, the host, and the destination device. The authentication token transfer procedure is performed similarly as the authentication token issue procedure, particularly at the destination device which displays identical behavior.

When the user provides to the host the authentication token, the secret for the token, and the new secret to replace the existing secret at the destination device, the host relays the token transfer request in a similar manner as in token issue request, and receives a challenge from the destination device. Then the host delivers the challenge along with the three-piece information provided by the user to the source device, which in turn verifies the token using the secret to authenticate the user and checks if the user-provided token is indeed a group authentication token. Then the source device generates a response to the destination device's challenge in an

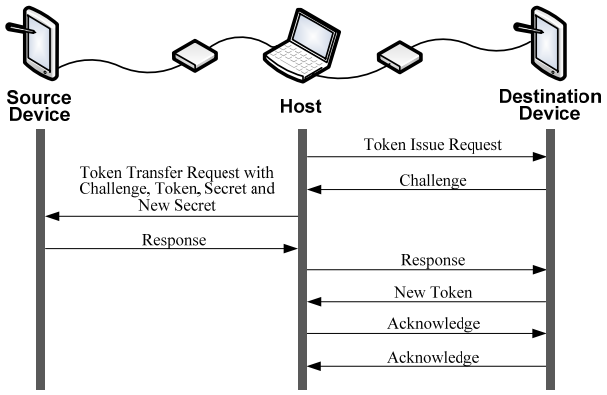


Fig. 2. Authentication Token Transfer Procedure.

identical manner as the server generates a response to a device's challenge in the authentication token issue procedure. The MNAA value and access level information of the existing token is preserved in the transferred token, but the secret is replaced by the new secret provided by the user. TI corresponding to the existing token, stored in the source device's non-volatile storage, is deleted in order to discard the token and prevent its further use. Lastly, the response generated by the source device is delivered, and is verified at the destination device which thereafter issues an authentication token for the host then stores the TI in its non-volatile storage.

4. Access Control

As devices differ in security level requirements depending on their type or features, debug mechanisms must also vary accordingly in their test functionalities depending on the type of users or the lifecycle of their target devices. For example, device developers involved in earlier development stages such as chip circuitry tests or board-level interconnection tests require different sets of test functionalities from the maintenance engineers working with devices in the field. The proposed approach in this paper provides a fine-grained access control mechanism which reflects the variability of security requirements.

The proposed approach classifies devices based on their security sensitivity into multiple security categories. A smartphone, for example, is differentiated from a generic portable media player because it stores secret keys for firmware or software authentication. Devices can employ different levels of protection based on the user's level of clearance granted by the server, managing each user's device access using the access level information found within the user's authentication token. The access level can be an hierarchical security level classified by the security sensitivity of the device resource made available through the debug port, or a non-hierarchical security level classified by feature sets necessitated by different user roles on a need-to-know basis.

This paper's approach proposes an access level assignment scheme based on roles, in which accessible device categories and access levels on those devices are managed according to the role performed by the individual user. For instance, the smartphone board-level interconnection tester role will have access to identical device categories but different access levels from the firmware developer role. If a role requires sequential access to multiple devices, group token ownership can be authorized for the role.

During the authentication token issue procedure, the server authenticates a user and grants the appropriate access level according to the user's role. The target device's serial number is used to check if the user is requesting access to a device belonging to the device category permitted for the user's role. A user can have authentication tokens with varying access levels depending on the user's current role.

5. Architecture

The architecture of the proposed approach is as shown in Fig. 3. The device communicates with the host locally through the debug tool and uses the network when communicating with the server. Besides the debug tool, the host runs separate authentication software for processing protocols. The authentication software provides the user interface, controls the overall execution of protocols, relays communications between the device and the server, and manages the device's protocol executions by using the debug tool software to read from or write to the device the protocol instructions or the authentication information.

In this paper's design, the device includes an authentication module and an access controller, both implemented in hardware to prevent modification. These components are structurally independent from the

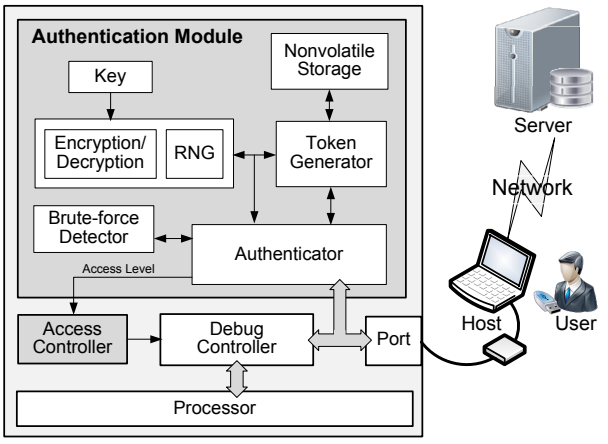


Fig. 3. Architecture of the Proposed Approach.

device's processor or its bus to eliminate the possibility of influence from any software-based attacks or malfunctions, and only accept control from the host through the debug port.

The authentication module is a functional block which receives the host's instructions through the debug port and executes certain steps of the protocols defined herein. The module also generates and verifies challenges, responses, and authentication tokens. In case a possible brute-force attack is detected (such as when the number of consecutive authentication failures by a user exceeds a predefined threshold value), the authentication module deletes the token's corresponding TI to disable its further use. TI is stored in a non-volatile storage by the authentication module. Memory space requirement for TI storage is practically insignificant as a token's TI has very small memory footprint, and typically only a small number of tokens is expected to be in use at any single time. Assigning a dedicated storage for TI is recommended for enhanced security. However, in case a larger memory has to be provided for TI storage to facilitate a large number of users, or if dedicated storage is too costly to implement, then TI storage can be assigned to a portion of an existing on-chip non-volatile storage on the device. In this case, hardware-based access control for the TI storage portion to restrict read/write operations other than by the authentication module is mandatory. In [13, 14] a secure bus architecture example is provided to illustrate the implementation of a hardware-based control.

The access controller determines the user's access level based on the authentication result. In the initial state where the user is not yet authenticated, the access controller controls the debug controller's actions to restrict all test functions except the serial number transfer function. After the user successfully completes authentication, the authentication module delivers the user's access level information to the access controller and the debug controller is controlled to allow tests permitted by the user's given access level. An example of access controller implementation to fine-control scan chain operations based on access levels is provided in [15].

IV. SECURITY PROTOCOL

In this Chapter we define the security protocols for supporting the execution of authentication token issue procedure, user authentication procedure, and authentication token transfer procedure as defined by the proposed approach. Notations used in this Chapter are summarized in Table 1.

Table 1. Notation

Notation	Description
UN	User name for user authentication by the server
PW	User password a paired with *UN*
SN	Unique identification number of a device
T	Authentication token
TI	Token generation information of *T*
TS	Secret of *T* provided by a user
TN	Unique identification number of *T*
AL	Device access level assigned to a user
G	Group token ownership of a user
MNAA	Maximum number of authentications allowed using *T*
C_D	Challenge generated by a device
R_S	Response by the server in response to C_D
R_D	Response by a device in response to C_D
N_C	Random number generated by a device for C_D
N_T	Random number generated by a device for *T*
K	Secret key shared between a device and the server
$E_x(y)$	Symmetric key encryption (x: key, y: text)
$D_x(y)$	Symmetric key decryption (x: key, y: text)
||	String concatenation

1. Assumptions

All proposed protocols in this paper use symmetric encryption to ensure security, which requires the device and the server to share a secret key *K*. All devices are

assigned a unique serial number *SN* for device identification, where serial number assignment schematics may vary depending on the security policy in effect or for the purpose of management efficiency. If an identical key is assigned to a particular device group, a 'group' field can be employed as part of the serial number schematics. The server is assumed to store the *SN* and *K* pair for all devices, along with their corresponding user account information, in its database.

$$Database = \begin{Bmatrix} Device-list : (SN_1 : K_1, ..., SN_n : K_n) \\ User-list : (UN_1 : PW_1, ..., UN_n : PW_n) \end{Bmatrix}$$

Each authentication token *T* includes the following information: the maximum number of authentications allowed using the token (*MNAA*), the user's access level on the debug port (*AL*), group token ownership of the user (*G*), and the unique token serial number (*TN*). An issued token *T* is classified using its corresponding device's serial number *SN* and is stored at the token storage.

$$Token\ Storage = \{T-list : (SN_1 : T_1, ..., SN_n : T_n)\}$$

The authentication module of the device issues *T* and stores the resulting token generation information *TI* within the non-volatile storage of the device after categorizing them using *TN*.

$$Nonvolatile\ Storage = \{TI\text{-}list : (TN_1 : TI_1, ..., TN_n : TI_n)\}$$

2. Authentication Token Issue Protocol

The authentication token issue protocol defines a secure 16-stage procedure for issuing an authentication token, as shown in Fig.4. Details for individual steps are as follows:

Step (1) : The user inputs to the host the user name *UN*, password for the user name *PW*, and the secret *TS* to be used for authentication token generation at the device, and requests for authentication.

Step (2) : The host, after establishing a connection with the device, obtains the device's *SN*. If *SN* has been obtained in prior or is available through different means, then this step can be omitted.

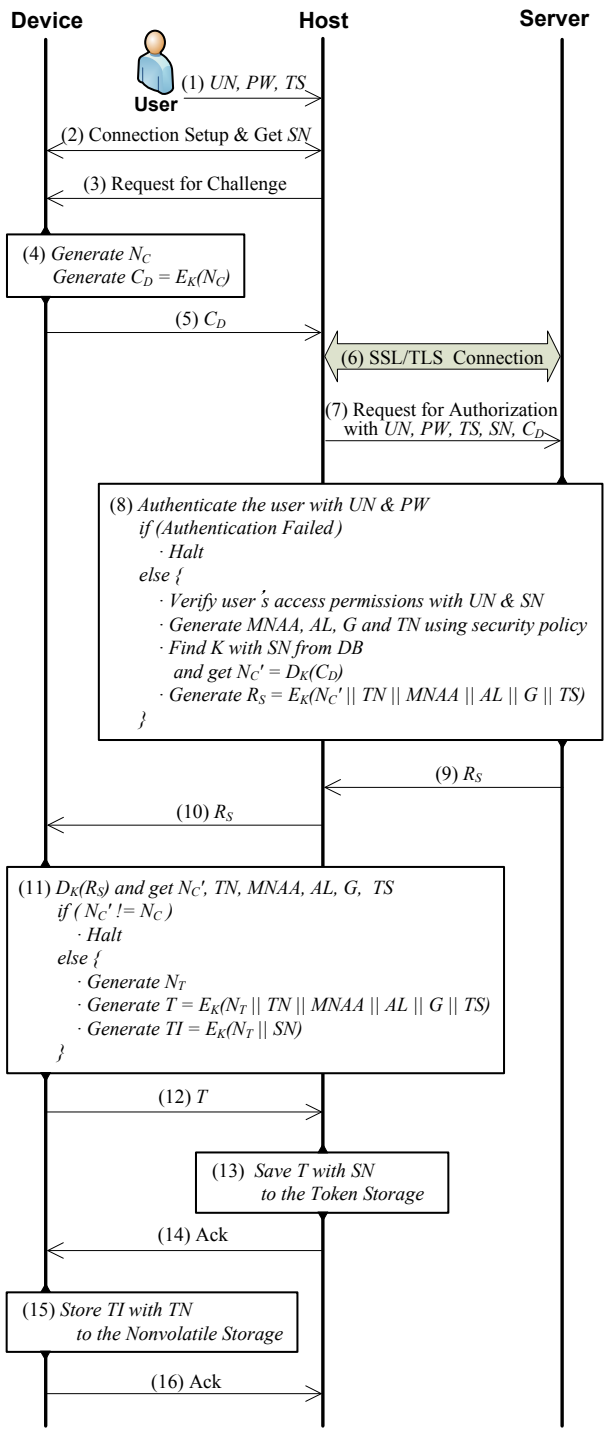


Fig. 4. Authentication Token Issue Protocol.

Step (3)~(5) : the host requests the device to issue an authentication token. The device first generates a random number N_C, then encrypts it using the shared key *K* to generate the challenge $C_D=E_K(N_C)$. This challenge is subsequently delivered to the host.

Step (6)~(7) : To ensure communication security, the host establishes an SSL/TLS connection with the server. Then the host sends the user-provided *UN*, *PW* and *TS* to the server, along with the device-provided *SN* and C_D, to request for user authentication and authorization.

Step (8)~(9) : In these steps the server authenticates the user and generates a response which provides the user's authorization information. First, the server authenticates the user with *UN* and *PW* delivered through the host. Then the server uses *UN* and *SN* to check the user's access permissions on the target device against the security policy in effect. Upon successful authentication and permission check, the server generates *MNAA*, *AL* and *TN* for the user in correspondence to the pertaining security policy, and also assigns group token ownership *G* if desired by an authorized user. Next, the server generates a response which includes the user's authorization information. Using *SN* to fetch the shared key *K* from the database, the server decrypts the challenge C_D to obtain the device-generated N_C'. Then all user authorization information (*MNAA*, *AL* and *G*) are concatenated with the information used in token generation (*TS* and *TN*) and are encrypted using *K* to generate the response $R_S = E_K(N_C' \,||\, TN \,||\, MNAA \,||\, AL \,||\, G \,||\, TS)$. The resulting response is sent to the host as a response to the device's challenge C_D.

Step (10) : The host delivers the server's response R_S to the device as a response to the device-generated C_D.

Step (11)~(12) : In these steps, the device verifies R_S and then generates the authentication token *T* and its corresponding token generation information *TI*. First, R_S is decrypted using *K* to obtain N_C', which is compared with N_C from step (4) to confirm that R_S has not been corrupted during transmission and to check that the user is correctly authenticated and authorized by an authentic server. Then the device generates a new random number N_T, concatenates it with *MNAA*, *AL*, *G*, *TN* and *TS* all obtained from R_S, and encrypts them to generate $T = E_K(N_T \,||\, TN \,||\, MNAA \,||\, AL \,||\, G \,||\, TS)$. Following on, the device encrypts N_T and *SN* to generate $TI = E_K(N_T \,||\, SN)$, which is the corresponding token generation information for *T*. Finally, *T* is sent to the host.

Step (13)~(14) : The host stores the *T* and *SN* pair sent by the device within its token storage, and acknowledges the device of the result.

Step (15)~(16) : If all previous steps have been successful, the device stores the *TI* and *TN* pair within its non-volatile storage, acknowledges the host of the successful result, and terminates the authentication token issue procedure.

3. User Authentication Protocol

The user authentication protocol is used only after an authentication token for a device has been successfully issued. Therefore we can safely assume that the user already possesses a device-generated *T* and that the device has stored its corresponding *TI* stored within the non-volatile storage. The protocol follows 12 steps as shown in Fig. 5, with details for each step as follows:

Step (1) : The user inputs to the host the authentication token *T* and its secret *TS*, and requests for authentication. In case the user wants to replace the secret of *T* with a new secret during the user authentication procedure, the user also inputs new secret *NewTS* to the host.

Step (2)~(4) : The host establishes a connection with the device and obtains the device's serial number *SN*. After confirming that user-input *T* is generated by the target device through matching it with *SN*, the host sends user authentication request to the device along with *T* and *TS*. *NewTS* is sent to the device if available.

Step (5) : In this step, the device authenticates the user using the host-delivered *T* and *TS*, along with *TI* which is generated during authentication token issue procedure. Then the device updates *MNAA* and re-generates *T* with new *MNAA*. First, in user authentication, *T* is decrypted using *K* to obtain *TS'* which is compared with the user-provided *TS* to confirm that the user is the owner of the token. Then *TN*, the identification number of *T*, is used to fetch *T*'s generation information *TI* from the non-volatile storage. *TI* is decrypted using *K* to obtain N_T' and *SN'*, which are in turn compared respectively with N_T and *SN*. If both comparisons match, authenticity of *T* is verified. Thereafter, *MNAA* value in *T* is reduced by 1 to get *MNAA'*. If *MNAA'* is still 1 or greater after the reduction, a new random number N_T is generated in order to re-generate *T* and *TI* with *MNAA'*.

Step (6) : The device notifies the host of the authentication result and at the same time informs the remaining number of authentications left by sending *MNAA'* value to the host. If the value of *MNAA'* is equal

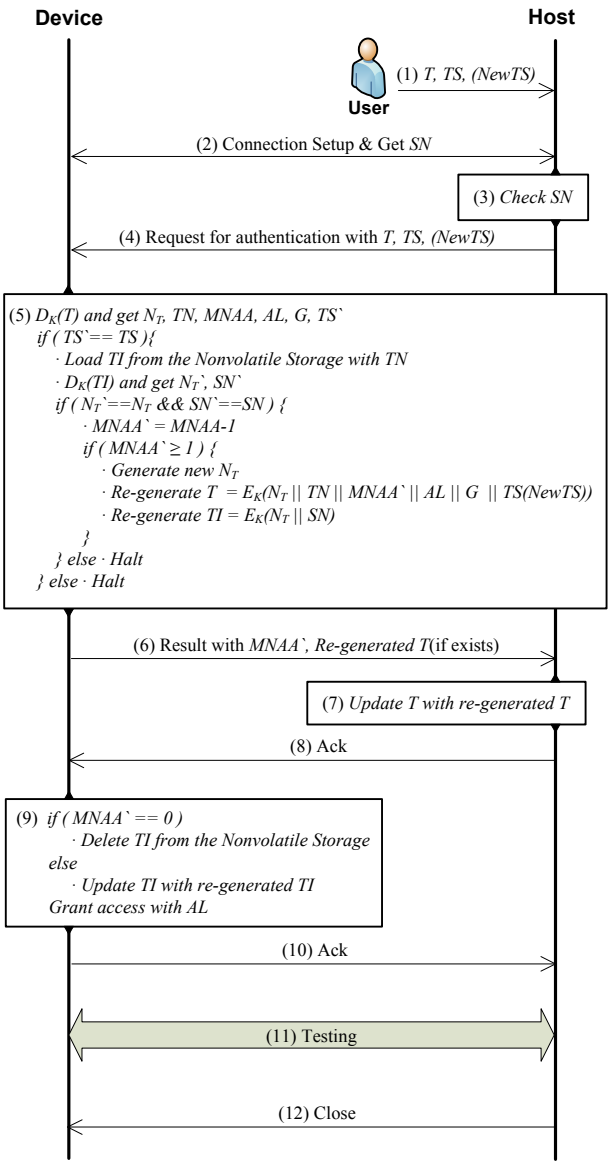


Fig. 5. User Authentication Protocol.

to or greater than 1, then newly generated *T* is also delivered to the host.

Step (7)~(8) : The host acknowledges the device's authentication result notification. If new *T* has been sent by the device, then the user's *T* is replaced with the new *T*.

Step (9)~(10) : Upon receiving the host's acknowledgement, the device either removes *TI* from the non-volatile storage if *MNAA'* value equals zero, or replaces the existing *TI* with the re-generated *TI* if *MNAA'* value is 1 or greater. Debug port is then opened to enable user access appropriate for the user's access level *AL*, and an acknowledgement is sent back to the host to signal the initiation of a test.

Step (11)~(12) : The user performs tests using the opened debug port features. After all tests are complete, the device is notified to close the opened port.

4. Authentication Token Transfer Protocol

The authentication token transfer protocol defines a secure 16-step procedure for transferring a group authentication token between devices. Here we assume that the *T* owned by the user already has *G* which authorizes it to be transferred to other devices. In this protocol the destination device behaves identically as the target device in the authentication token issue protocol. Details for individual steps are as follows:

Step (1) : The user inputs to the host *T*, *TS*, and a new secret *NewTS* to be used at the destination device, and requests for a token transfer.

Step (2)~(5) : The host establishes a connection with the destination device, requests for an authentication token to be generated on the device, and receives a challenge C_D.

Step (6)~(7) : The host establishes a connection with the source device and delivers *T*, *TS*, *NewTS*, and C_D to the device. Then the host requests for authentication token transfer.

Step (8)~(9) : In these steps, the source device verifies the authentication token delivered by the host, generates R_D in response to C_D, and sends it to the host. First, the source device decrypts *T* delivered by the host using *K* to obtain N_T, *TN*, *MNAA*, *AL*, *G*, and *TS'*. *TS'* is compared with the user-provided *TS* to verify the user. Then *TI* is fetched from the non-volatile storage using *TN*, and is decrypted using *K* to obtain N_T' and *SN'*. N_T' and *SN'* is respectively compared with the previously-obtained N_T and device-provided *SN* to verify the authentication token. If all previous steps have been successful, *G* is checked to confirm that *T* is transferrable to other devices. If *T* is confirmed as a group authentication token, R_D is generated in response to C_D in the identical manner as in step 8 of the authentication token issue protocol. While existing *TN*, *MNAA*, *AL*, and *G* values in *T* are reused, *T*'s secret is replaced with *NewTS*. After R_D is successfully generated, it is delivered to the host.

Step (10)~(12) : The host sends R_D generated by the source device to the destination device. Upon receiving

R_D, the destination device verifies the response in the identical manner as in step 11 of the authentication token issue protocol, then generates a new T and its corresponding TI. The newly generated T is delivered to the host.

Step (13)~(16) : After the new T is stored in the host, the destination device stores TI in its storage and completes the token transfer procedure.

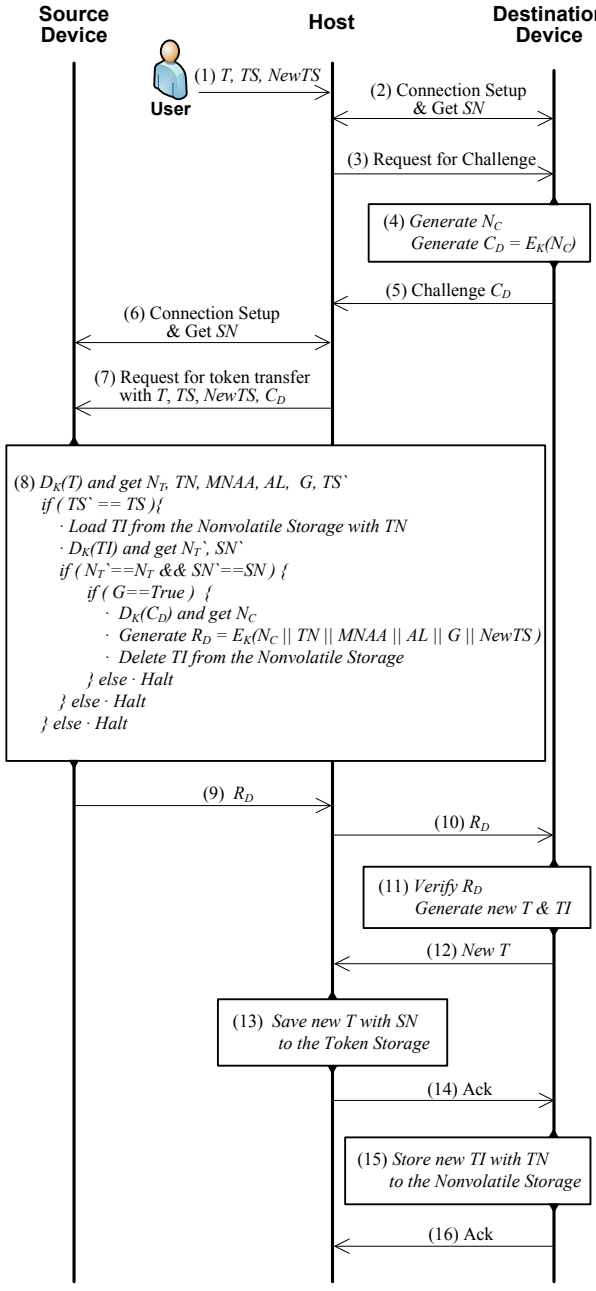


Fig. 6. Authentication Token Transfer rotocol.

V. SECURITY ANALYSIS

In this Chapter we analyze the security of the protocols proposed by this paper's debug port protection mechanism. The server is assumed to maintain its security through independent mechanisms. Physical attack to the device with the purpose of key value analysis is left out of scope of this paper. Notations used in this Chapter are identical to the previous Chapter.

1. Authentication Token Issue Protocol

The authentication token issue protocol utilizes SSL/TLS communication, which supports mutual authentication between the host and the server while ensuring data integrity and confidentiality. Between the device and the server, symmetric encryption using shared secret key K and random number N_C generated by the device ensures authenticity, integrity and confidentiality of transferred authentication messages. The combination of security mechanisms employed in the authentication token issue protocol effectively protects against various attacks aimed to obtain authentication tokens. For instance, an attacker incapable of authorizing oneself through the server may attempt to deceive the device by submitting fraud information. To have the device generate an authentication token based on fraud information, the attacker needs to be able to arbitrarily generate R_S which correctly corresponds to device-generated C_D and includes user-provided TS. However, without the secret key K, the attacker is incapable of decrypting C_D and thereby cannot arbitrarily generate a valid R_S. Therefore it is impossible for an authorized attacker to deceive the device with fraud information. Even authenticated users are restricted from arbitrarily modifying $MNAA$, G, AL, and TN values within server-generated R_S due to the same limitation. In another case, a previously authenticated user can store some information from a previously successful authentication token generation event, with hopes to attempt a replay attack using the stored information at a later time. However, such attacks are also foiled by the security mechanisms of the token issue protocol. An authentic user can collect T, C_D, R_S, and TS during a token issue. However, in order for the user to deceive the device by posing as the server, the device must generate a C_D which corresponds to the

previously stored information. The random number N_C generated for each C_D in every authentication token issue attempt effectively prevents a prospective replay attacker from deceiving the device using stored information.

2. User Authentication Protocol

The user authentication procedure can be assumed to be secure as it is executed only in a local environment where the device and the host is directly connected. Authentication tokens, on the other hand, are stored within the host and are subject to misplacement or theft, introducing openings for attackers to use counterfeited tokens to attempt access. However, such attack is ineffective in the proposed approach. During step 5 of the user authentication protocol, the device decrypts *T* delivered by the host to obtain *TS'* and compares it with the host-delivered *TS* to verify the user. Thereafter, N_T' obtained by decrypting *TI* is compared with N_T within *T* while *SN'* is compared to the device's own *SN* in order to check if *T* provided by the user is indeed the token that the device itself has issued. An attacker without *TS* is incapable of authenticating himself, and even with *TS*, he cannot reuse a *T* issued by another device. Even if an attacker has a previously legitimate token issued by the device which is now expired, the device employs a policy of replacing *TI* value after each successful authentication and removing *TI* when a token's *MNAA* reaches zero, which can effectively counter such attacks.

3. Authentication Token Transfer Protocol

The authentication token transfer protocol is generally similar to the authentication token issue protocol and shares its security characteristics. Both protocols ensure confidentiality and integrity of messages sent between the source device and the destination device: confidentiality by using the shared secret key *K*, and integrity by employing the random number N_C generated by the destination device. These mechanisms make it impractical for an attacker to intervene during a token transfer between devices. Even if an attacker attempts to interfere with a token transfer using a compromised host, it will still be impossible to arbitrarily modify the *AL* and *MNAA* value assigned to an authentication token. Further, the source device completely deletes *TI* in order to invalidate the user's current token at step 8 of the authentication token transfer protocol before delivering R_D to the destination device. Therefore a user is unable to reuse a token to authenticate oneself if it has already been transferred to another device.

VI. COMPARISON WITH RELATED WORKS

The proposed approach in this paper incorporates the strengths of previous works while alleviating their weaknesses, improving both security and availability in a better balance between the two traits. As shown in Table 2, the proposed approach supports application of fine-grained security policies through the server based on various user and device information in a similar fashion as existing three-entity protocols. A user may be assigned different access levels depending on the time of authentication, role assigned to the user, and the type of device to be authenticated for. Unique to the proposed approach is the feature to assign a limit to the number of authentications permitted using a particular token. A token with no authentications left are automatically discarded by the device. This feature enhances the level of security in comparison to [12], in which no control is exerted over the usage of server-issued credential, practically offering unlimited access after the initial issue. As in [12], the proposed approach does not require server access after the user has initially issued a token. In addition, the proposed approach enables the user to test multiple devices using a single token, and supports changing of the secret of the token without server access. These new features further enhance the availability of debug ports in various test environments. For instance, when a user needs to use more than one device for a task, the user had to issue separate credentials for each device individually through the server if the test environment utilized existing protocols. With the proposed approach, however, the user can request for a group authentication token which grants access to all the devices designated to the same group. In this way, devices in the field without network access can be tested by issuing group authentication tokens for them in advance and then transferring them to the devices later. In another case where a group of users use a device in turn, each user needs to have issued his individual authentication token for the device. The proposed approach alleviates such

Table 2. Comparison of Security, Availability and Efficiency against Previous Works

	R.F.Buskey [11]	K.Y.Park [12]	Proposed Method
Fine-grained security policy control for user & device	Yes	Yes	Yes
Control over user's access level	Yes	Yes	Yes
Control over user's number of authentications	N/A	No	Yes
User authentication without server access	No	Yes	Yes
Multiple device testing without server access	No	No	Yes
Token(credential)'s secret(pwd) change without server	N/A	No	Yes
†Device operations count for token(credential) generation	N/A	4H + 9X + R	4E + D + 2R
†Device operations count for user authentication	6M + 2R	3H + 5X + R	2E + 2D + R

† X: Exclusive OR operation, H: Cryptographic hash operation, R: Random nonce generation, E: Symmetric key encryption, D: Symmetric key decryption, M: Scalar multiplication

inefficiency by enabling users to share the token's secret for a single authentication token which grants access to all users of the group. Unlike in [12], the proposed approach lets the secret of a token be changed locally by the user any time. If the shared secret has been compromised or changes are made to membership of the group, the group can simply change the existing token's secret on the device without communicating with the server to issue a new token. Then, after the group's task is finished, the token can simply be discarded to prevent further use. With the proposed features, security can be maintained above a certain level even in a test environment where no network is accessible and collaboration with unreliable partner is required.

The proposed approach does not demand higher operational complexity on the device even though it offers improved security and availability over previous protocols. Specifically, the proposed approach employs relatively simple and easy-to-implement operations in comparison to [11] which require complex scalar multiplications or [12] which repeatedly nests hash and XOR operations to a degree hardware controller design is made difficult due to complex operation order. Further, the repetitive patterns found in the three protocols of the proposed approach facilitate resource reuse and enable simpler design of the hardware authentication IP block on the device. The time required to authenticate users accessing debug ports can be regarded insignificant, taking into account the fact that simultaneous user authentication is not possible and that authentication does not introduce additional latency into the test process afterwards.

VII. IMPLEMENTATION

The proposed approach of this paper is implemented on a debugging environment based on the IEEE 1149.1 standard. The host's authentication software communicates with the server via SSL/TLS, while communication with the device is through the debug tool. Trace32 manufactured by Louterbach was used for the debug tool in our implementation. The authentication software establishes a socket-based communication with the Trace32 software and controls the authentication module's behavior through standard test instructions.

Specification for the authentication module used for experimentation is as shown in Table 3. The authentication module employs a 128-bit key AES-CBC symmetric encryption algorithm and a 160-bit random number generator. For TI storage, a 256-byte area within the on-chip flash memory has been allocated and access to this area is hardware-controlled to only permit read and write access by the authentication module. For the purpose of implementing the access controller, the scan register cells within the device are organized into groups based on the security sensitivity and relevancy of the corresponding

Table 3. Implementation Summary

Component	Length
AES Encryption Key	128-bit
Random Number	160-bit
Token	384-bit (*MNAA*:12, *AL*:8, *G*:1, N_D:160, *TN*:32, *S*:160 and *Padding bits*)
Token Generation Info.	256-bit (N_T:160, *SN*:48 and *Padding bits*)

Table 4. Implementation Result

Component	Slices
Authentication Module	1366-Slices (AES Core: 941, RNG: 192, other: 233)
Access Controller	21-Slices
Debug Controller	52-Slices
Total	1439-Slices

device components, and only the register cells permitted access by the token's access level are activated.

The authentication module and the access controller are designed using Verilog HDL, synthesized with Synopsys Sinplify Premier v9.62 and verified of operation accuracy on Xilinx FPGA Spartan-3AN. A total of 1439 slices are used for implementation (excluding the nonvolatile storage) as shown in Table 4. Synthesis using Synopsys Design Compiler with Samsung Electronics 45 nm process technology yielded approximately 28k gates, which is a competitive result in context of enhancing security small embedded devices, for example mobile devices, in which area overhead is an important factor [16]. Further, the proposed approach does not incur time over during the test procedure other than the time necessary for authentication itself.

VIII. CONCLUSIONS

In this paper we have proposed a debug port protection mechanism to prevent device hacking. As debug ports exist for supporting device tests and maintenance functions, their availability must be ensured even after necessary security measures have been applied. Device access by hackers should be prevented, while authorized users should be permitted access all test functions necessitated by their roles. The proposed approach improves on security and availability over existing methods through user authentication using tokens. After a user issues a token for the device necessary to perform one's role, the token can be used to authenticate oneself on a device to use its debug port anywhere, anytime. If a group token has been issued, the token can be used to authenticate the user on multiple devices in turn, thereby ensuring high level of availability in various test environments. Each token stores its owner's access level information and the maximum number of authentications allowed using the token, which respectively controls debug port access and prevents unchecked use of a token. This enables us to use tokens to assign appropriate security levels according to the role and authorizations of a user. The proposed approach is inexpensive to implement and is unrestricted by the application environment, facilitating flexible adoption to various environments in demand of debug port security or hardware interface security.

ACKNOWLEDGMENTS

This research was supported by a research grant from Samsung Electronics for constructing a trusted mobile platform.

REFERENCES

[1] P. Kocher, et al, "Security as a new dimension in embedded system design," *Proceedings of Design Automation Conference* (DAC), pp.753-760, Jun., 2004.

[2] OMTP Hardware Working Group, "OMTP Security Threats on Embedded Consumer Devices," *Open Mobile Terminal Platform*, May, 2009.

[3] R. Kapur, "Security vs. test quality: are they mutually exclusive?," *Proceedings of International Test Conference* (ITC), pp.1414, Oct., 2004.

[4] Institute of Electrical and Electronic Engineers, "Standard test access port and boundary-scan architecture," *IEEE.Std. 1149.1,* 2001.

[5] M. F. Breeuwsma, "Forensic imaging of embedded systems using JTAG (boundary-scan)," *Journal of Digital Investigation*, Vol.3, No.1, pp.32-42, Mar., 2006.

[6] B. Yang, K. Wu, R. Karri, "Secure scan: a design-for-test architecture for crypto chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.25, No.10, pp.2287-2293, Oct., 2006.

[7] Y. Liu, K. Wu, R. Karri, "Scan-based attacks on linear feedback shift register based stream ciphers," *ACM Transactions on Design Automation of Electronic Systems*, Vol.16, No.2, Article No.20, Mar., 2011.

[8] A. Ashkenazi, D. Akselrod, "Platform independent overall security architecture in multi-processor

system-on-chip integrated circuits for use in mobile phones and handheld devices," *Computer & Electrical Engineering*, Vol.33, No.5-6, pp.407-424, May, 2007.

[9] F. Novak, A. Biasizzo, "Security extension for IEEE std 1149.1," *Journal of Electronic Testing: Theory and Applications*, Vol.22, No.3, pp.301-303, Jun., 2006.

[10] J. Lee, et el, "Securing scan design using lock & key technique," *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems* (DFT), pp.51-62, Oct., 2005.

[11] R. F. Buskey, B. B. Frosik, "Protected JTAG," *Proceedings of International Conference on Parallel Processing Workshops* (ICPPW), pp.405-414, Aug., 2006.

[12] K. Y. Park, et el "JTAG Security System Based on Credentials," *Journal of Electronic Testing: Theory and Applications*, Vol.26, No.5, pp.549-557, Oct., 2010.

[13] L. W. Kim, J. D. Villasenor, "A System-On-Chip Bus Architecture for Thwarting Integrated Circuit Trojan Horses," *IEEE Transactions on Very Large Scale Integration Systems*, Vol.19, No.10, pp.1921-1926, Oct., 2011.

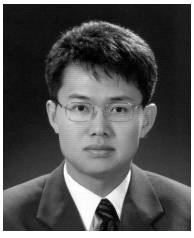
[14] C. Lee, "Smart Bus Arbiter for QoS control in H.264 decoders," *Journal of Semiconductor Technology and Science*,Vol.11, No.1, pp.33-39, Mar., 2011.

[15] L. Pierce, S. Tragoudas, "Multi-Level Secure JTAG Architecture," *Proceedings of International On-Line Testing Symposium* (IOLTS), pp.208-209, Jul., 2011.

[16] J. Kim, S. Han, R. Jewell, "Timing Analysis Techniques Review for sub-30 nm Circuit Designs," *Journal of Semiconductor Technology and Science*,Vol.10, No.4, pp.292-299, Dec., 2010.

Keun-Young Park received the B.S and M.S degree from the Department of Computer Science and Engineering, Sogang University, Seoul, Korea, in 2007 and 2009 respectively. He is currently pursuing the Ph.D. degree at the Department of Computer Science and Engineering, Sogang University. His interests include mobile security, cryptography, and system-on-a-chip technology. He is a CISA, and CISSP.

Sang-Guun Yoo received the B.S degree from the Faculty of Computer System Engineering, Army Polytechnic School, in 2002, and the M.S degree from the Department of Computer Science and Engineering, Sogang University, Seoul, Korea, in 2010. He is one of the co-founders of ExtremoSoftware (Microsoft Gold Certified Partner), where worked as software architect from 2001 to 2005. From 2005 to 2007, He collaborated as a research member and professor in the Department of Computer Science and Multimedia, International University of Ecuador. From 2006 to 2007, He also worked as an IT consultant for the Army Intelligence Agency of Ecuador.



Juho Kim received the B.S. and Ph.D. degrees in Computer and Information Science from the University of Minnesota in 1987 and 1995 respectively. He then worked as a senior member of technical staff at Cadence Design Systems until 1997. Professor Kim joined the Department of Computer Science and Engineering, Sogang University, Seoul, Korea, in 1997, and was department chair from 2005 to 2007.