Implementing the Secure Protocol for Exchanging the Symmetric Key of FPGA-based Embedded Systems

Tran Thanh[1], Tran Hoang Vu[1], Nguyen Van Cuong[2], Pham Ngoc Nam[1]
[1]School of Electronics and Telecommunications, Hanoi University of Science and Technology, Hanoi, Vietnam.
Email:{thanh.tran, vu.tranhoang, nam.phamngoc}@hust.edu.vn
[2] Faculty of Electronics and Telecommunications, Danang University of Science and Technology, Danang, Vietnam.
Email: nvcuong2000@gmail.com

***Abstract* - Cryptographic solution for protecting data which pass through an insecure public network is widely applied. To ensure the data confidentiality and availability, the secret key must be exchanged securely between parties before beginning a transaction session. This paper presents a protocol to enhance the flexibility and secrecy of symmetric key exchange over the Internet. Our approach uses an asymmetric encryption algorithm to protect symmetric encryption keys from thefts and tampers over a transmission line. In addition, this paper presents a protocol to ensure the integrity, confidentiality of the symmetric key, and the freshness of a transaction session. Experimental results from a prototype system based on FPGA are also revealed.**

Keywords – Security key, symmetric key, security algorithm.

I. INTRODUCTION

Cryptography is the practice and study of techniques for secure communication in the presence of third parties. Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The main classical cipher types are transposition and substitution ciphers. Modern cryptography is heavily based on mathematical theory and computer science practice. Cryptographic algorithms are designed too hard to break in practice for any attackers. Along with a key, they are used in the encryption and decryption of data.

Cryptographic algorithms are classified into two main groups, including symmetric encryption (also called symmetric key encryption or secret key encryption) and asymmetric encryption (also called public key encryption), as shown in Fig. 1. When using symmetric encryption algorithms, both parties share the same key for encryption and decryption. To provide privacy, this key needs to be kept confidential. Once somebody else gets to know the key, it is not safe any more. A few well-known examples are: DES[1], Triple-DES[1], AES[2], BlowFish[3]. On the other hand, asymmetric encryption algorithms use pairs of keys, among which, one is used for encryption and the other for decryption. Typically, the decryption key is kept secretly, therefore called "secret key" or "private key". Meanwhile, the encryption key is spread to all who might want to send encrypted data, called "public key". The secret key cannot be reconstructed from the public key. Well-known asymmetric key encryption algorithms are RSA[4], DSA[5].

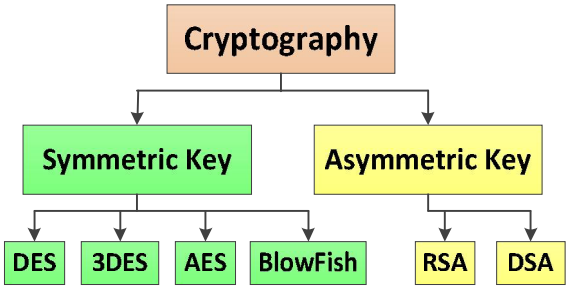

Figure 1. Overview of the type of cryptography

The security of the encrypted data (called ciphertext) in modern cryptography depends on two aspects: The strength of the encryption algorithm and confidentiality of the key. More generally, cryptographic algorithms have been standardized and published. Therefore, the key is one of the important elements of security processes that should be kept

confidential. Using asymmetric algorithms, the secret key must not be shared. Every user only needs to keep one secret key in secrecy and a collection of public keysthat can be published. With symmetric algorithms, every pair of users would need to have an own shared secret key. Finally, the only symmetric key should be kept in confidentiality and has to be shared or exchanged, as shown in Fig. 2.

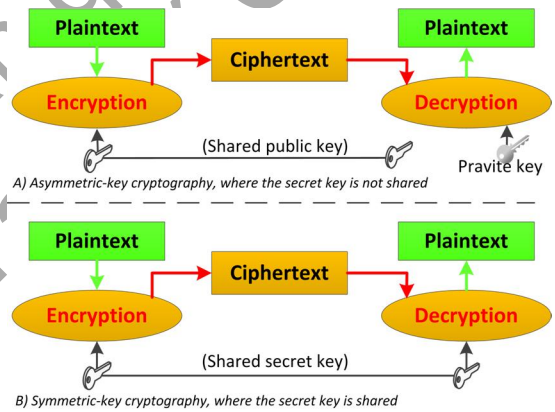

Figure 2. Sharing key of the type of cryptography

Asymmetric algorithm seem to be ideally suitable for real-world use: As the secret key does not have to be shared, the risk of getting known is rather low than symmetric algorithm. However, as presented in [6][7], asymmetric algorithms are much slower than symmetric ones. Therefore, secure communication uses symmetric keys for bulk message encryption, while asymmetric keys are used for symmetric key exchange and digital signatures.

Exchanging symmetric keys in untrusted environment would pose potential risks such as theft, malicious, man-in-middle attacker, etc. Serveral works have been carried out this issue. Santosh Deshpande proposed a protocol for secure key exchange[8]. His approach supports preventing Denial of Service (DoS) attacks, but the freshness and continuity of a transmission session have not considered. Alternatively, Saar Drimer presented a protocol for secure remote update of FPGA-based system [9]. In that his work, he used K_{UL} for MAC calculation and data encryption, but he didn't present how the secure key to exchange between both the sides.

Basing on the above considerations, this paper presents a scheme combining the symmetric, asymmetric and hash algorithms to ensure flexibility and securely exchanging the secret keys over the Internet. Our proposed protocol is improved from Saar Drimer's protocol for exchanging the secure key. In this protocol, we describes the work including a system designer, who buys IP cores from IP vendors and a user, who updates his/her system from service providers. Moreover, in our protocol, the parameters for implementing the security of a partially reconfigurable embedded system are defined and added. The proposed protocol is referenced to Internet Key Exchange protocol in [10],[11].

The rest of the paper is organized as follows. Section II describes our proposed scheme. The scheme is demonstrated by a prototype system based on Xilinx Spartan-6 LX45 FPGA Atlys board in Section III. Conclusions are drawn in Section IV.

II. THE PROPOSED SCHEME

A. Building the protocol

To implement the proposed scheme, we consider exchanging the secure key of IP cores, which transfer over the Internet between a System Integrator (SysInt) and an IP Vendor (IPVend) or a SysInter and a User. The parties and the implementation process are shown in Fig. 3.

System Integrator (SysInt) designs the FPGA system and provides it to the user. SysInt has physical access to the product and can issue a product upgrade in the field. A typical system consists of custom elements and multiple third-party (IPVend) IP cores. IP cores can be distributed in various formats: HDL sources, netlists or FPGA device-specific partial bitstreams, depending on the level of trust between SysInt and IPVend.

IP Vendor (IPVend) provides reusable components (IP cores), and related data sheet, or may design an IP block to meet a provided subsystem specification. IPVend is typically not directly involved in the system level FPGA design process. IPVend wishes to protect its own design secrets.

Trusted Authority (TAut) is an authorization and/or certification center. TAut confirms the key generation process during initial system start-up and certifies the resulting key material. TAut is assumed to be trustworthy by all parties and not involved in the system development process.

User is an end-customer who operates the system, possibly in hostile environments. User requires the system to be secure, but could also try to gain personal profit by attempting to circumvent the implemented security countermeasures.

In the proposed scheme, at each session, participants must use the same authentication algorithms (e.g., SHA-512), symmetric encryption algorithms (e.g., AES-256) and asymmetric encryption algorithms (e.g., RSA). Parameters in the proposed protocols have been stored in the profile database of SysInt, IPVend, TAut and User side, and are listed below.

N_{Sys}:	Nonce generated by SysInt
SK_S:	SysInt's Symmetric Key
SSK_S:	SysInt's Session Symmetric Key
PK_V:	IPVend's Public Key
SK_V:	IPVend's Secret (Private) Key
PK_U:	User's Public Key
S_{ID}:	SysInt Identifier
V_{ID}:	IPVend Identifier
TA_{ID}:	TAut Identifier
U_{ID}:	User Identifier
M_x, M_x':	HMAC values

1) *Keys are authenticated by the TAut*

This is the highest safety level of exchanging keys. The keys of parties are always authenticated by the Trusted Authority before making the transaction. The details will be described in the following steps:

- Beginning a transaction session, SysInt generates a number only use once N_{Sys}. The N_{Sys}, S_{ID}, V_{ID} and "ReqIP" will be sent to IPVend.
- IPVend receives and verifies parameters. If these parameters are available, IPVend will send the public key PK_V to SysInt.

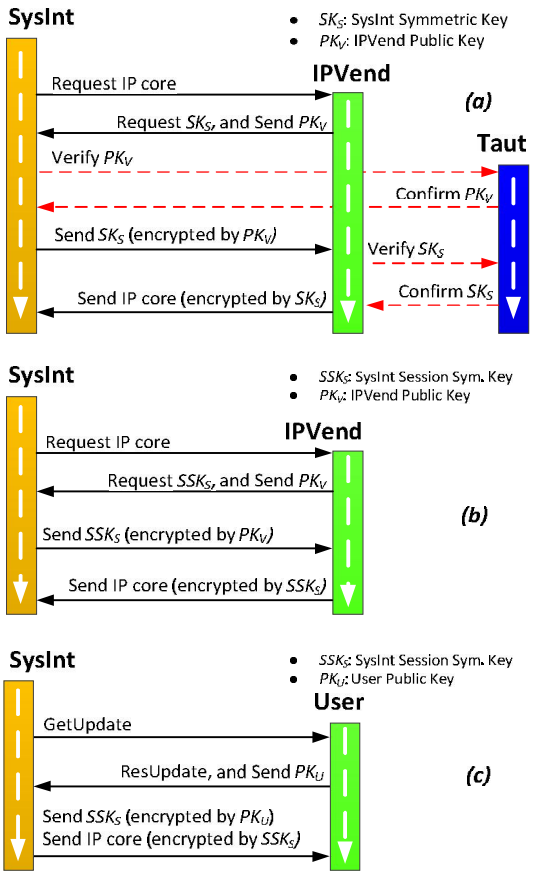

Figure 3. Protocol of the secure key exchange: (a) Keys are authenticated; (b),(c) Keys are not authenticated

- SysInt receives and sends the PK_V with IPVend ID to the TAut to authenticate. If the PK_V is true, SysInt will use this key to encrypt SK_S, and then send the encrypted SK_S to IPVend.
- In turn, IPVend sends the encrypted SK_S with SysInt ID to Taut to authenticate.
- TAut uses IPVend's PK_V to encrypt SysInt's SK_S stored in a database. If this result is same the encrypted SK_S which have been received from IPVend, TAut will send a "ConfirmSK_S" command to IPVend in true.
- IPVend uses an own private key SK_V to decrypt the encrypted SK_S, and then uses the SK_S to encrypt IP core and transfers encryptedIP to SysInt.

At each session, both the PK_V and SK_S are always exchanged and authenticated. The SK_V did not exchange. The PK_V does not need to be kept secret. Only the SK_S needs to be kept in confidentiality ; and it had been encrypted by the PK_V before passing through the Internet as mentioned above Fig. 3(a). Algorithms of the secure key exchange are described in more detail in the pseudo code form, and shown as follows:

Algorithm A1: System Integrator

S1. Generate(N_{Sys})
S2. M_0 = HMAC("ReqIP", S_{ID}, V_{ID}, N_{Sys})
S3. Send("ReqIP", S_{ID}, V_{ID}, N_{Sys}, M_0)
S4. Receive("ReqSK_S", PK_V, M_1)
S5. M_1' = HMAC("ReqSK_S", PK_V, M_0)
S6. If $M_1' \neq M_1$ then **goto S1**
S7. M_2 = HMAC("VerifyPK_V", S_{ID}, V_{ID}, TA_{ID}, PK_V, M_1)
S8. Send("VerifyPK_V", S_{ID}, V_{ID}, TA_{ID}, PK_V, M_1, M_2)
S9. Receive("ConfirmPK_V", M_3)
S10. M_3' = HMAC("ConfirmPK_V", M_2)
S11. If ("ConfirmPK_V " ≠ PK_V_OK) or $M_3' \neq M_3$ then **goto S1**
S12. M_4 = HMAC(EncryptedSK_S, M_1)
S13. Send(EncryptedSK_S, M_4)
S14. Receive(EncryptedIP, M_7)

Algorithm A2: IP Vendor

V1. Receive("ReqIP", S_{ID}, V_{ID}, N_{Sys}, M_0)
V2. M_0' = HMAC("ReqIP", S_{ID}, V_{ID}, N_{Sys})
V3. If $M_0' \neq M_0$ then **goto V1**
V4. M_1 = HMAC("ReqSK_S", PK_V, M_0)
V5. Send("ReqSK_S", PK_V, M_1)
V6. Receive(EncryptedSK_S, M_4)
V7. M_4' = HMAC(EncryptedSK_S, M_1)
V8. If $M_4' \neq M_4$ then **goto V1**
V9. M_5 = HMAC("VerifySK_S", EncryptedSK_S, M_4)
V10. Send("VerifySK_S", EncryptedSK_S, S_{ID}, V_{ID}, PK_V, M_4, M_5)
V11. Receive("ConfirmSK_S", M_6)
V12. M_6' = HMAC("ConfirmSK_S", M_5)
V13. If ("ConfirmSK_S " ≠ SK_S_OK) or $M_6' \neq M_6$ then **goto V1**
V14. M_7 = HMAC(EncryptedIP, M_6)
V15. Send(EncryptedIP, M_7)

Algorithm A3: Trusted Authority

T1. Receive("VerifyPK_V", S_{ID}, V_{ID}, TA_{ID}, PK_V, M_1, M_2)
T2. M_2' = HMAC("VerifyPK_V", S_{ID}, V_{ID}, TA_{ID}, PK_V, M_1)
T3. If $M_2' \neq M_2$ then **goto T1**
T4. M_3 = HMAC("ConfirmPK_V", M_2)
T5. Send("ConfirmPK_V", M_3)
T6. Receive("VerifySK_S", EncryptSK_S, S_{ID}, V_{ID}, PK_V, M_4, M_5)
T7. M_5' = HMAC("VerifySK_S", EncryptedSK_S, M_4)
T8. If $M_5' \neq M_5$ then **goto T1**
T9. M_6 = HMAC("ConfirmSK_S", M_5)
T10. Send("ConfirmSK_S", M_6)

2) *Keys are authenticated by the TAut*

When the parties can trust each other, the key exchange does not need to be authenticated through TAut. However, to prevent risks, session symmetric key (herein called session key) is proposed for use. The session key is a different symmetric key that can be used for each exchange between partners. These keys are generated for each transaction to remove the requirement for maintenance of symmetric keys. Invalidation of compromised or expired symmetric keys is no longer a problem. Each session key is used once only with one message.

The algorithms of the secure key exchange between SysInt and IPVend or User are described as below, in which, the N_{Sys} and HMACs for the freshness and continuity of a transmission session is similar to case of having the above authentication. The session key SSK_S is generated by SysInt. SysInt uses the PK_V to encrypt this key, and exchanges to IPVend.

- *The algorithms of secure key exchange between SysInt and IPVend (Fig. 3(b)):*

Algorithm B1: System Integrator

S1. Generate(N_{Sys})
S2. M_0 = HMAC("ReqIP", S_{ID}, V_{ID}, N_{Sys})
S3. Send("ReqIP", S_{ID}, V_{ID}, N_{Sys}, M_0)
S4. Receive("ReqSSK_S", PK_V, M_1)
S5. M_1' = HMAC("ReqSSK_S", PK_V, M_0)
S6. If $M_1' \neq M_1$ then **goto S1**

S7. M_2 = HMAC(EncryptedSSK_S, M_1)
S8. Send(EncryptedSSK_S, M_2)
S9. Receive(EncryptedIP, M_3)

Algorithm B2: IP Vendor
V1. Receive("ReqIP", S_{ID}, V_{ID}, N_{Sys}, M_0)
V2. M_0' = HMAC("ReqIP", S_{ID}, V_{ID}, N_{Sys})
V3. If $M_0' \neq M_0$ then **goto V1**
V4. M_1 = HMAC("ReqSSK_S", PK_V, M_0)
V5. Send("ReqSSK_S", PK_V, M_1)
V6. Receive(EncryptedSSK_S, M_2)
V7. M_2' = HMAC(EncryptedSSK_S, M_1)
V8. If $M_2' \neq M_2$ then **goto V1**
V9. M_3 = HMAC(EncryptedIP, M_2)
V10. Send(EncryptedIP, M_3)

- *The algorithms of secure key exchange between SysInt and User (Fig. 3(c)):*

Algorithm C1: System Integrator
S1. Generate(N_{Sys})
S2. M_0 = HMAC("GetUpdate", S_{ID}, U_{ID}, N_{Sys})
S3. Send("GetUpdate", S_{ID}, U_{ID}, N_{Sys}, M_0)
S4. Receive("ReqSSK_S", PK_U, M_1)
S5. M_1' = HMAC("ReqSSK_S", PK_U, M_0)
S6. If $M_1' \neq M_1$ then **goto** S1
S7. M_2 = HMAC(EncryptedSSK_S, M_1)
S8. Send(EncryptedSSK_S, EncryptedIP, M_2)

Algorithm C2: User
U1. Receive("GetUpdate", S_{ID}, U_{ID}, N_{Sys}, M_0)
U2. M_0' = HMAC("GetUpdate", U_{ID}, U_{ID}, N_{Sys})
U3. If $M_0' \neq M_0$ then **goto U1**
U4. M_1 = HMAC("ReqSSK_S", PK_U, M_0)
U5. Send("ReqSSK_S", PK_U, M_1)
U6. Receive(EncryptedSSK_S, M_2)
U7. M_2' = HMAC(EncryptedSSK_S, EncryptedIP, M_1)
U8. If $M_2' \neq M_2$ then **goto U1**
U9. Receive (EncryptedIP)

B. Security analysis

The nonce N_{Sys} is generated by SysInt must be an unpredictable random number and also has no opportunity to repeat. This prevents attackers from replaying the data of the previous session. The N_{Sys} should be large enough to make the creation of a dictionary of responses that can be replayed impractically. S. Drimer *et.al.*[9] suggested that the use of an uniform distributed 64-bit word for N_{Sys} will ensure that an attacker who wants to perform 10^3 queries per second must spend a lot of time up to many decades to find its matching value.

HMAC values (M_0, M_0', M_1, M_2', etc.) are generated by the SHA-512 algorithm with 512-bit length. The HMACs provide brute-force upload attempts with an equal generous safe margin. The parameters of algorithms are used to calculate the HMAC known as M_0. Then M_0 is used again as a parameter to calculate the HMAC named as M_1. The process repeats several times until the end of the session. The HMACs are applied throughout the update process to prevent man-in-the-middle attackers to replay old bitstream or malicious code.

RSA asymmetric algorithm is used to protect symmetric key. RSA public keys are generated by multiplying large prime numbers together and are derived from factoring the product of the two these numbers. Deriving the private key from the known public key is very difficult. If prime numbers are given large enough, it is impossible to derive a private key from a public key. Moreover, the private key is not exchanged over the Internet, so leaked key risk is less probable, which means the symmetric key is always safely protected.

Measuring security of the RSA have presented in [12], Kefa Rabah shows that factoring the 129-digital number (426 bits) in 1994 required 5000 MIPS-year (MIPS means the million instructions per second; and one MIPS-year is equal to the number of instructions executed during one year of computing at one million instructions per second). It also used idle time on 1600 computers around the world over an eight-month period. When using RSA encryption algorithm to encrypt symmetric keys, the support of 512 bits to 1024 bits variable key length is required Besides, to protect very high value transactions - at least a 1024 bits or higher key should be used.

III. IMPLEMENTATION AND RESULTS

A. System setup

To test the proposed scheme, we have built a prototype system as in Fig. 4, consisting of a

reconfigurable embedded platform based on Xilinx Spartan-6 LX45 FPGA Atlys board which plays the role of IPVend or User and a laptop plays the role of SysInt. The FPGA Atlys board and the laptop are connected via a TCP/IP protocol.

Figure 4. Protocol of the secure key exchange

On the Xilinx Spartan-6 SC6SLX45 chip, we embedded a MicroBlaze soft-core microprocessor using the Xilinx Embedded Development Kit (EDK) ver. 14.1 software. EDK toolset allows designers to easily create platforms based on either MicroBlaze. EDK offers a variety of peripherals (UARTs, counter, Ethernet, memory controller, general-purpose I/O and so on) and a one-connection solution based on the structure of the IBM CoreConnect bus [13].

As analyzed and presented above, the protocol of secure key exchange using two SHA-512 and RSA algorithms. In particular, the SHA-512 algorithm is used to authenticate transaction sessions, secret keys and bulk messages (e.g., IP cores). Therefore, it can be built in hardware to speed up calculations or software to install and change flexibility. The RSA algorithm is only used for exchanging the symmetric key, so we recommend that it should be built in software to reduce the hardware resources cost of the system.

For software implementation, we used the open source code for RSA and SHA-512 and ported them on MicroBlaze with some modifications. However, the code had not been optimized yet. It will be a subject for future works.

In order to compare and analyze the efficiency of the systems, we implemented the RSA algorithm in embedded software running on MicroBlaze and SHA-512 algorithm in hardware. Results are shown in Table I, II.

B. Results and evaluation

We assumed that it is necessary to encrypt and decrypt for exchanging 256bit key length of AES-256 symmetric encryption algorithm, which means encoding and exchanging 32bytes capacity of data. Table I shows the costs of resource, and the measured execution time of the RSA encryptor at SysInt and RSA decryptor at IPVend, by using the pairs of public key *(N = 517, e = 3)* and private key *(N = 517, d = 307)*.

Table I. Implementation results of the RSA algorithm

RSA Core	LOC in C	Size in memory (KB)	Execution time
RSA_Decrypt	74	1.89 KB	0,9 s
RSA_Encrypt	72	1.83 KB	0,01 s

Table II shows the costs of resource and the throughput of the SHA-512 algorithm in both hardware and embedded software. As it can be seen in Table II, the throughput of software solution is two times slower than that of the hardware solution.

Table II. Resoures use of the SHA-512 algorithm

SHA-512 Core	Size in memory	Hardware Utilization *(Used / Available)*	Through-put
Hardware	-	Slices: 852/54576 LUTs: 2052/27288	30Mbps
Software	138 KB	-	15.2Mbps

The flexibility of the proposed scheme is the ability to combine both the RSA and SHA-512 algorithms in software or RSA algorithm in software and SHA-512 algorithm in hardware, which depends on the system resource and the specific application.

IV. CONCLUSIONS

The proposed method of security key exchange has combined 3 factors: a number only used once, the public encryption algorithm and hash function. This ensures the integrity and confidentiality of secret key passing through untrusted network as well as the originality and availability of keys for parties, and the freshness and continuity of transmission sessions.

In addition, based on the proposed method, system designers can implement algorithms in hardware or software flexibility. Future work will be optimizeing the performance of the algorithms that is implemented on FPGA-based reconfigurable embedded systems.

REFERENCES

[1] NIST., "FIPS 46-3: Data Encryption Standard (DES)," 2009.

[2] NIST., "FIPS 197: Advanced Encryption Standard (AES)," 2001.

[3] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Fast Software Encryption SE - 24*, R. Anderson, Ed. Springer Berlin Heidelberg, 1994, pp. 191–204.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] NIST., "FIPS 186-3: Digital Signature Standard (DSS)," 2009.

[6] Y. Kumar, R. Munjal, and H. Sharma, "Comparison of Symmetric and Asymmetric Cryptography with Existing Vulnerabilities and Countermeasures," *Int. J. Comput. Sci. Manag. Stud.*, vol. 11, no. 03, pp. 60–63, 2011.

[7] H. W. H. Wang, B. S. B. Sheng, C. C. Tan, and Q. L. Q. Li, "Comparing Symmetric-key and Public-key Based Security Schemes in Sensor Networks: A Case Study of User Access Control," in *2008 The 28th International Conference on Distributed Computing Systems*, 2008.

[8] S. Deshpande, "Symmetric Key Management: A new approach," *Int. J. Eng. Comput. Sci.*, vol. 1, no. 3, pp. 125–136, 2012.

[9] S. Drimer, "A protocol for secure remote updates of FPGA configurations," *Lect. Notes Comput. Sci.*, vol. 5453, pp. 50–61, 2009.

[10] A. Steffen, "Secure Communications in Embedded Systems," *CRC Ind. Inf. Technol. Handb.*, pp. 1–15, 2004.

[11] D. R. Kuhn, V. C. Hu, W. T. Polk, and C. Shu-Jen, "SP 800-32: Introduction to Public Key Technology and the Federal PKI Infrastructure," *National Institute of Standards and Technology*, no. February, pp. 1–54, 2001.

[12] R. Kefa, "Implementing Secure RSA Cryptosystens Using Your Own Cryptographic JCE Provider," *J. Aplpied Sci.*, vol. 6, no. 3, pp. 482–510, 2006.

[13] Xilinx, "UG081: Microblaze processor reference guide (Ver9.0)," *Xilinx, Inc.*, vol. 081, 2006.

AUTHORS' BIOGRAPHIES

Tran Thanh is a Ph.D. student in Electrical Engineering in ESRC laboratory of Hanoi University of Science and Technology (Vietnam), where he has been since 2010. He has a B. Eng. degree in Electronics and Telecommunications from Danang University of Technology and a M.Sc. degree from the University of Danang in 1995 and 2007, respectively. He works at The Vietnam Research Institute of Electronics, Informatics and Automation. His research is in reconfigurable computing, reconfigurable embedded systems and FPGA security.

Tran Hoang Vu received B. Eng. degree in Electronics and Telecommunications from Da Nang University of Technology and M.Sc. degree from the University of Danang (Vietnam) in 2004 and 2007, respectively. From 2004 until now he has been working at Danang College of Technology-The University of Danang, Vietnam. His research interests include Reducing power consumption of Data Center Networks, reconfigurable embedded systems and low-power embedded system design.

Nguyen Van Cuong received B. Sc. degree In Solid State Electronics from Hue University of Science (Vietnam) in 1987. He was awarded a Ph.D. degree in Electrical Engineering from Uni. BW Munich, Germany in 2000. From 2000 until now he has been working at Danang University of Science and Technology, Vietnam. His main disciplinary focus is on embedded systems and energy–aware VLSI system design.

Pham Ngoc Nam received B. Eng. degree In electronics and Telecommunications from Hanoi University of Science and Technology (Vietnam) and M.Sc. degree in Artificial Intelligence from K.U. Leuven (Belgium) in 1997 and 1999, respectively. He was awarded a Ph.D. degree in Electrical Engineering from K.U.Leuven in 2004. From 2004 until now he has been working at Hanoi University of Science and Technology, Vietnam. His research interests include reconfigurable embedded systems and low-power embedded system design.