Hardware-Based Protection for Data Security at Run-Time on Embedded Systems

Xiang Wang[a], Xiaobing Zhang[b], Weike Wang, Pei Du, Zhun Zhang, Yuntong Tian, Qiang Hao and Bin Xu

School of Electronic and Information Engineering, Beihang University, Beijing 100191, China
[a,b] Corresponding author: wxiang@buaa.edu.cn; cassy@buaa.edu.cn

Abstract. The security of embedded systems has attracted much attention as they are being used in more and more fields. The rapid growth and pervasive use of embedded systems make it easier for a sophisticated attacker to gain physical access to launch physical attacks on insecure off-chip memory and bus. This paper presents a novel hardware-based security mechanism to protect confidentiality and integrity of data, preventing the system data from being stolen or tampered by a malicious attacker. The proposed mechanism protects the confidentiality of data using advanced encryption standard (AES) stream encryption algorithm in parallel with the memory access process. This mechanism provides integrity protection for data by attaching integrity signatures generated using hash algorithm to data stored in external memory. The signature is verified when data is fetched into the chip. The security architecture has been tested and validated on the system on a programmable chip (SoPC) with OR1200(processor based on OpenRISC1000 architecture) processor. The experimental result shows that the proposed security mechanism ensures the integrity and confidentiality of system data, introducing low performance penalties.

1. Introduction

The rapid development of microelectronics and computer technology has enabled embedded systems to be widely used in all spheres of our lives. Embedded systems play an increasingly important role in the current people's daily life, which are indispensable to modern communication devices, medical equipment, consumer electronics, home appliances, transportation systems, and even weapons systems, economy and so on. If a certain embedded system is attacked by the intruder, enhance huge losses. Therefore, the security of embedded systems is required by people in these applications. There are higher security requirements in some areas, such as military and economy. Consequently, security becomes a critical issue in embedded computer systems design and operation.

Defences for application code and control-flow against security exploits have been studied extensively [1]. However, few schemes have been presented to protect the security of program data since it is significantly harder than protecting code owing to the highly dynamic of data during the entire length of execution. The security of data is a very important part of the security of embedded system. The rapid growth and pervasive use of embedded systems makes it easier for a sophisticated attacker to gain physical access to launch physical attacks on insecure off-chip memory. With the help of advanced electronic equipment, an attacker can control the address/data bus to tap and tamper and inject or replay memory blocks when the program data are loading to the processor, resulting to leakage of confidential information or destruction of data.

In this paper, we propose a hardware-based architecture to protect the confidentiality and integrity of data at run-time. In order to prevent the attacker from stealing the confidential or sensitive information of the system, the data entering the untrusted area is encrypted, protecting the confidentiality of which. During program execution, the data is signed and verified integrity to prevent the system data from being corrupted or tampered with, the integrity of which protected. Any unauthorized change of data will be detected. To counter performance overheads induced by encryption and signature verification latency, the proposed architecture incorporates the following architectural enhancements: parallel encryption and decryption with memory access, parallelizable signatures and conditional data validation. Memory overhead due to signatures is reduced by protecting a block of data instead of a data transmitted with a single signature and encrypting signatures and storing them off-chip.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the threat model. Section 4 presents the security architecture in detail. Section 5 shows the experimental result and analysis. Section 6 concludes this paper.

2. Related Work

The security protection methods of embedded systems have attracted much attention among researchers and various techniques have been proposed to solve this problem. The early traditional methods are software-based and nowadays most technologies proposed are hardware assisted. Hardware assisted technologies are more efficient with a high processing speed and a small resource overhead. In this section, the related hardware-based techniques are examined as below.

Suh and his colleagues [2] propose the artificially expanded genetic information systems (AEGIS)secure processor. They introduce physical unclonable functions (PUFs) to generate the secrets needed by their architecture. A ground-breaking model for monolithic security is proposed. However, their architecture needs extensive operating system and compiler support. Yan et al. [3] describe a sign-and-verify architecture using Galois/Counter Mode cryptography. They protect dynamic data using split sequence numbers to reduce memory overhead and reduce the probability of a sequence number rollover. A tree-like structure is used to protect dynamic data against replay attacks. Multiple access to tree nodes results in a large delay. Gelbart [4] presents an architectural support for securing application data integrity. Their scheme can protect the application data from physical attacks. A method based on Advanced Encryption Standard - Galois/Counter Mode (AES-GCM) to protect the confidentiality and integrity of data is proposed by Vaslin et al [5]. In this method, each block of data serves as input to the AES-GCM. The generated ciphertext is written to an out-of-chip memory, and the resulting tag is stored in the chip. This scheme has advantages in terms of speed and security, but the on-chip storage overhead is high.

In [6], the author presents a hardware/software approach to secure the application data. Their work enhances the memory hierarchy to represent the attributes of each datum as security tags, and adds a configurable hardware checker that interprets the semantics of the tags and enforces the desired security policies. In [7], Hong et al. presents a cost-effective tag generation design (CETD). Unlike other existing schemes, where the tag generation logic is fixed and the related high implementation cost can hardly be reduced, this design offers flexibility for varied security levels. However, the data tag generated in CETD has certain correlation with the data itself, with a high tag collision rate. Liu Tao et al. [8] proposes an improved memory data label generation method for embedded processors. They present an enhancement which adds randomness to the input data with the bit flip and the non-linear Galois Field multiplication (GFM) operations, to safeguard the design against the integrity attack with any chosen values.

3. Threat Model

In this section, we will illustrate three classes of attacks to which computer systems may be subjected and the model of attacks on system data considered in this paper.

Security attacks can be classified into software attacks, which are launched through malicious software or by exploiting weaknesses in the software executing on the system, physical attacks, which operate via intrusive physical access into a system’s internals, and side-channel attacks, which are

based on observation of a system's implementation properties such as execution time or power consumption [9].

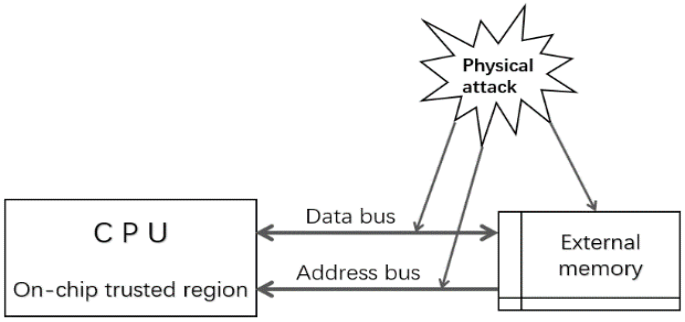

Figure 1. Threat model of system data

The kind of attacks considered in this paper is the physical attacks, as shown in the Fig. 1. We assume that the CPU is trusted, but any external device except for the processor is not trusted, for instance, the external memory and the data bus, which could be attacked. When the attacker obtains physical access to the embedded equipment, he can interfere with the external memory or the communication between CPU and memory to launch attacks as below.

(1) Data spoofing attacks: The attacker can modify the system data in the external memory with a random value or inject a portion of data as shown in the Fig. 2(a).

(2) Data splicing attacks: A splicing attack involves the attacker intercepting a bus request and returning a valid but non-requested block as shown in the Fig. 2(b).

(3) Data replay attacks: The attacker can record a portion of data. Then he intercepts a bus request and returns an old, potentially stale version of the requested block. As shown in the Fig. 2(c).

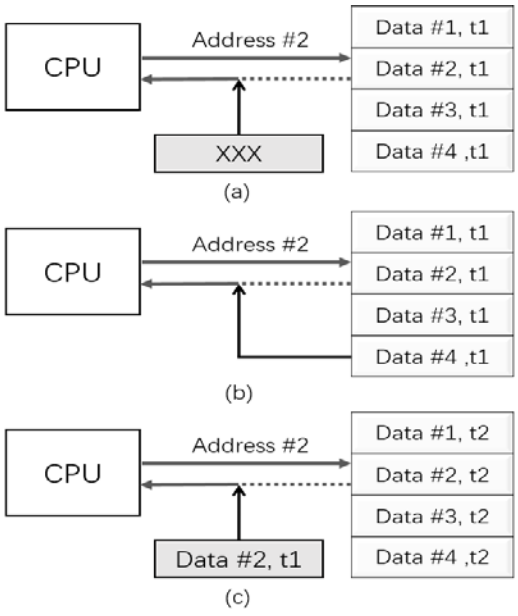

Figure 2. Spoofing attacks, splicing attacks and replay attacks

4. Security Architecture

This section describes the proposed hardware-based data protection architecture and the implementation and working process of the proposal. The proposed architecture protects the confidentiality and integrity of the system data.

4.1. Overview

Fig. 3 provides the overview of our designed hardware-based technique that can protect the confidentiality and integrity of data dynamically at run-time.

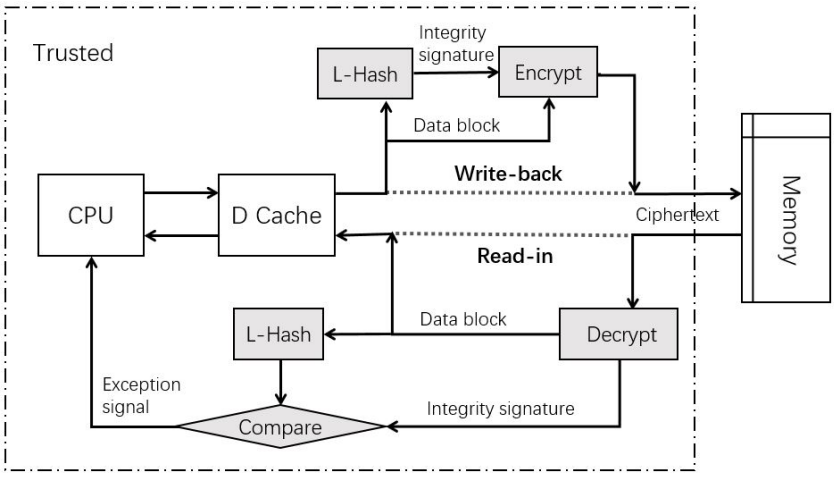

Figure 3. Hardware protection mechanism

In the proposed security mechanism, lightweight hash algorithm(L-Hash) [10] is used to sign and verify data stored in external memory with its hash value, protecting the integrity of the system data and advanced encryption standard (AES) stream encryption algorithm is used to encrypt and decrypt both data and its integrity signature, protecting the confidentiality of them. Then, we consider the size of data protection granularity, which is a trade-off between security and system performance overhead. In this security mechanism, on-chip data are signed and encrypted when they are evicted out, and the data are decrypted and verified after they are fetched from external memory. Data blocks are transmitted between external memory and processor when the data cache misses. Consequently, the most suitable protected data block size is the cache line size of the lowest level of the data cache or some multiple thereof, or the size of the fetch buffer in systems without a cache. Without loss of generality, in the rest of this paper we focus on a system with separate data and instruction first level caches and no second level cache. We take the cache line size as the size of the protected data block, which not only guarantees security, but also simplifies the process of data protection.

Data cache is located in the on-chip trusted area on which cached content is considered immune to attack. In the designed scheme, we add a hardware-based security protection module between the data cache and external memory, which in the on-chip trusted region. When the data cache conflicts and writes back a data block to the external memory, the data block is attached with a signature calculated using lightweight hash algorithm, that records its integrity information. Then the data block and its signature are encrypted using stream encryption, and encrypted ciphertext is stored in external memory. When the data cache misses and fetches a data block from the external memory, the data block and its signature are decrypted first. The hash value of the decrypted data block is calculated again and compared with decrypted signature. When a mismatch is detected, the security module will send an exception signal to the CPU, which triggers the response mechanism.

4.2. Data Integrity

Integrity attack refers to a class of attacks that affect the normal execution of system programs by destroying or tampering with system data and program code. Integrity is violated whenever any unauthorized code is executed or unauthorized data is used. Integrity protection mainly depends on various authentication algorithms. In this paper, the implementation of the data integrity protection architecture is based on the lightweight hash (L-Hash) algorithm, which is used to generate integrity signatures and verify the signatures of data blocks stored in external memory.

Hash algorithm is a kind of unidirectional compression algorithm, whose mechanism is to compress the information of any length into a fixed length output through a certain compression

structure, and the output is called hash value. L-hash algorithm improves the traditional hash algorithm and makes a trade-off between security, speed, energy consumption and implementation cost. The safety boundary of the L-Hash algorithm based on the sponge structure can be given by the following formula[11]:

$$Collision\ resistance: min\left\{2^{\frac{n}{2}}, 2^{\frac{c}{2}}\right\} \quad (1)$$

$$Second-preimage\ resistance: min\left\{2^{n}, 2^{\frac{c}{2}}\right\} \quad (2)$$

$$Preimage\ resistance: min\left\{2^{n}, 2^{c}, max\left\{2^{n-r}, 2^{\frac{c}{2}}\right\}\right\} \quad (3)$$

Several optional parameters and corresponding complexity security boundaries are given in the table 1. Taking our experimental platform as an example, the OR1200 processor used is an open-source soft core processor with a 32-bit data bus. Consequently, the version with a packet length of 32 bits is adopted, which affects the security of the algorithm in a certain extent but can improve the information processing speed.

Table 1: Security Boundaries with Different Parameters

n	b	c	r	Collision	2nd-preimage	Preimage
80	96	80	16	2^{40}	2^{40}	2^{64}
96	96	80	16	2^{40}	2^{40}	2^{80}
96	**96**	**64**	**32**	$\mathbf{2^{32}}$	$\mathbf{2^{32}}$	$\mathbf{2^{64}}$
128	128	120	8	2^{60}	2^{60}	2^{96}
128	128	120	8	2^{60}	2^{60}	2^{120}

Data integrity is ensured by verifying the integrity signature of the data at run time. Different from the code which can be determined after compiling and linking, the data is highly dynamic and can be modified during the running of the program. For data integrity protection, not only spoofing attacks and splicing attacks need to be considered, but also replay attacks should be considered further. To prevent these attacks at the same time, the integrity signature of the data is a cryptographic function of the following: (a) the actual value of the data, (b) the starting virtual address of the data, (c) a number that can mark the order of time. The value of the data is necessary to prevent spoofing attacks and partial splicing attacks and replay attacks. Because once the value of the data is tampered with, the integrity validation information that contains the data content will change. Using the data's address prevents splicing attacks, since data residing at different addresses will have different signatures. To prevent replay attacks on data, a number that can mark the order of time is required to ensure that all fetched dynamic data is up-to-date. We call the number a timestamp, which is associated with a protected block of data.

Next, we discuss how to generate the timestamp. To prevent being stolen by a malicious attacker, it needs to be generated in the chip and stored in the trusted region within the chip. In the designed scheme, we use a counter to generate a count value as the timestamp to mark the time. Whenever the data cache writes back a data block to an external memory, the count value increases by one. Both the counter and timestamp need to be stored in trusted areas, and the timestamp is mapped to the corresponding data block by address. When the data block is read in, the corresponding timestamp is retrieved in the timestamp memory according to the address.

4.3. Data Confidentiality

Confidential attacks are also very common attacks against data, aimed at stealing confidential and sensitive information from the system. The protection of data confidentiality is usually realized by encrypting the data in the untrusted area, which makes the data monitored or stolen by the attacker as incomprehensible random code, preventing the disclosure of the confidential information[12]. Consequently, the protection of data confidentiality depends on the encryption algorithm. In this paper, we adopt AES algorithm, which has the advantages of good security and easy hardware implementation.

There are two ways to encrypt data using AES algorithm. One is block encryption method, in which the content of the data block is used as the input of the encryption algorithm and the output is the ciphertext of the data block. The other is stream encryption, that is, the AES algorithm is used as a key stream generator to generate a series of pseudo-random numbers, and the data can be encrypted and decrypted by performing XOR operation with the sequence. Compared with the two methods, the latter has two outstanding advantages.

First, the stream encryption mechanism adds a small additional delay. To illustrate the process of encryption, we take our experimental platform as an example, which is a system on a programmable chip (SoPC) based on the OR1200 processor. The OR1200 processor is an open-source soft core processor with a 32-bit architecture and 32-bit separate data and address bus. The size of the data block transmitted between the data cache and external memory is 128 bits. Fig. 4 compares the delay caused by different encryption mechanisms. To complete the data block transmission on the bus, the data bus needs to be accessed four times. Block encryption requires the acquisition of a complete block of data before encryption and decryption. Compared with the two methods, the stream encryption increases the delay by less. In the proposed system, stream encryption method is used, in which the delay is mainly due to the calculation time of AES algorithm.

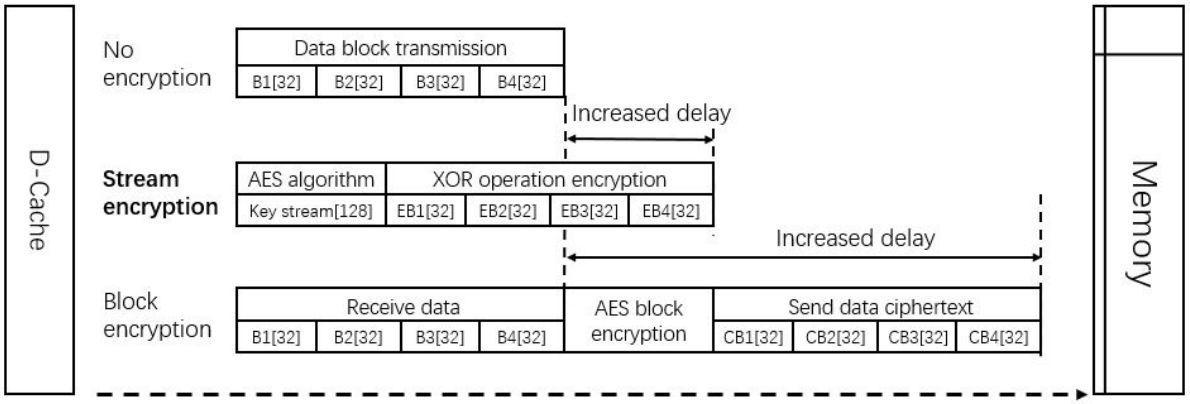

Figure 4. The delay caused by different encryption mechanisms

In addition, the stream encryption mechanism has relatively low hardware overhead. This is illustrated in Fig. 5. The encryption and decryption of stream encryption are implemented using the same hardware module as shown in the Fig. 5(a), which includes a key stream generation section and a simple XOR operation section. The encryption and decryption of block encryption are implemented by two separate modules as shown in the Fig. 5(b). Compared with the two methods, the stream encryption has less hardware overhead.

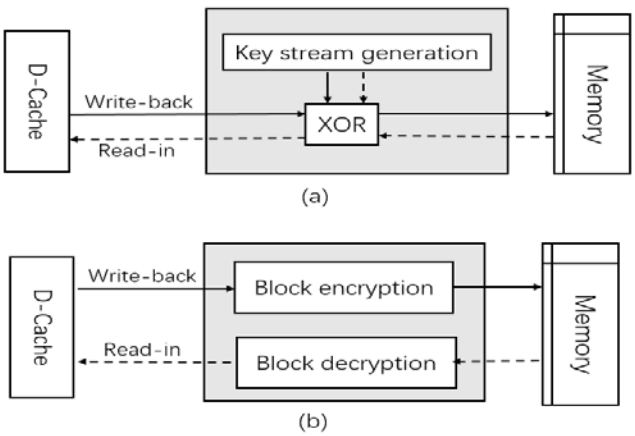

Figure 5. Hardware implementation

The process of encrypting and decrypting data using AES stream encryption can be expressed as the following:

$$C = P \oplus keystr = P \oplus AES_{\text{key}}(seed) \quad (4)$$

$$P = C \oplus keystr = C \oplus AES_{\text{key}}(seed) \quad (5)$$

Where C is the ciphertext, P is the plaintext data value, and *keystr* is the key stream having the same bit width as P, which is a pseudorandom number stream generated by using AES algorithm. In the formula of generating the key stream, key is the key of the AES algorithm and seed is the input, which uniquely corresponds to the plaintext. Operationally, when P is sent off chip, (4) is used; when C is read from memory, (5) is used. Calculating AES_{key} (seed) is carried while the processor is waiting for the memory. And XOR operation is performed during the transmission of a data stream. In this design, on-chip data are encrypted when they are evicted out due to cache conflicts. The data are stored in an encrypted format outside the on-chip trusted region. The decryption is carried out after the data are fetched from memory and before they are used by the CPU.

To ensure the security of stream encryption, it is significant for the key stream to be random and unique. If the key stream has a higher repetition rate, this is vulnerable even with encryption, and it may potentially permit an attack that doesn't take much effort. Consequently, the input of AES algorithm generating the key stream is also required good uniqueness. Next, we discuss how to select the key seeds. First of all, we have to make sure that the data at different locations in the external memory has different seeds. To that end, we use the physical address of the data block as part of the seed to ensure the spatial uniqueness of the key stream. In addition, for a specific address, the data written each time is required to correspond to a different seed. Therefore, we require that the seed contain time information to ensure the time uniqueness of the key stream. We can use the timestamp contained in the integrity signature as part of the seed.

5. Experimental Results

In terms of platform building, the embedded processor adopted is OR1200 which is a 32-bit scalar RISC with Harvard micro architecture. The cross-compiling tool for OR1200 is the popular and free GNU. The systematic simulation toolset is OR1KSim. The SoPC platform is built on a Xilinx FPGA.

5.1. Hardware Overhead

The SoPC platform is verified on a Xilinx virtex5 FPGA. Hardware resource consumption is shown in Table 2.

Table 2. Platform resource consumption

Slice Logic Utilization	Platform used	Security module used
Slice	2449	869
Slice	7400	1461
occupied	2864	536
Block	17	1
Total	540	18

5.2. Performance Penalty

In this paper, a mechanism for supporting data confidentiality and integrity is designed with little latency by performing the requisite cryptographic operations in parallel with memory accesses. The AES stream encryption is adopted to encrypt and decrypt data blocks, and the key stream is generated while waiting for the memory to answer, which needs 12 clock cycles. After the key stream is obtained, the data stream is transmitted, and it is encrypted through a simple XOR operation with the key stream during the transmission process. For the integrity of data, hash algorithm is used to sign and verify data blocks, and it takes 6 clock cycles to get hash values that contain data block content,

address, and time information. The computation of the hash algorithm is parallel to the transmission of the data stream, so there is no extra delay. In addition, the encrypted signature is stored in the off-chip memory following the data stream, and the signature transmission on the data bus requires 6 clock cycles. In conclusion, the performance penalty caused by the security module is mainly due to the delay caused by the generation of key stream and the transmission of integrity signature and it is relatively low.

5.3. Security and Overhead Trade-offs

The uniqueness of the key seed is very critical to the security of encryption. The timestamp is used to ensure the time uniqueness of seeds, so the size of the counter that produces timestamps is a problem worth considering. If the counter is too small, the count value will overflow quickly resulting in a high repetition rate of timestamp, and the time uniqueness of the key seed cannot be guaranteed well, so that the security of data encryption will be reduced. However, if the counter is too large, the storage of timestamps will take up too much on-chip storage space. Therefore, the size of the counter should be selected according to the application requirements, and there is a balance between security and storage overhead. In this paper, an 8-bit counter is used on our experimental platform. Each 128-bit data block of the system is attached with an 8-bit timestamp, with a 6.25% on-chip storage overhead.

6. Conclusions

This paper presents hardware security extensions suitable for implementation in embedded processors. The proposed architecture relies on the lightweight hash algorithm to protect the integrity of the system data by integrity signature, and AES stream encryption algorithm is used to protect the confidentiality of data dynamically at run-time. Cryptography computations are carried in parallel with memory accesses, minimizing the performance penalty. The experimental results show that the designed architecture can defend a wide range of common physical attacks, such as spoofing attack, splicing attack and replay attack, protecting the confidentiality and integrity of data effectively with low performance and hardware overhead.

7. Acknowledgments

This research is supported by the Key Project of National Science Foundation of China (Grant No. 61232009), the National Science Foundation of China (Grant No. 60973106, and No. 81571142), National High-tech R&D Project of China (863 Grant No. 2011AA010404).

8. References

[1] W. Wang, M. Liu, P. Du, Z Zhao, Y Tian, Q Hao, X Wang. *IEEE International Conference on Software Security and Assurance*, pp. 116-120 (2017)
[2] GE. Suh, CW. Odonnell, I. Sachdev, S. *Devadas. International Symposium on Computer Architecture*, **33** (2), pp. 25-36 (2005)
[3] C. Yan, D. Englender, M. Prvulovic, et al. *International Symposium on Computer Architecture,* **34** (2), pp. 179-190 (2006)
[4] O. Gelbart, E. Leontie, B. Narahari, R. Simha. *IEEE International Conference on Electro/Information Technology*, pp. 19-24 (2008)
[5] J. Crenne, R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, D. Unnikrishnan. ACM T EMBED COMPUT S, 12 (3), pp. 1-23 (2013)
[6] D. Arora , S. Ravi , A. Raghunathan, K. Jha Niraj. IEEE T VLSI SYST, 15 (5), pp. 546-559 (2007)
[7] M. Hong, H. Guo, SX. Hu. International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 17–26 (2012)
[8] T. Liu, H. Guo, S. Parameswaran, XS. Hu. INTEGRATION, **56**, pp. 96-104 (2017)
[9] S J. Crane, S. Volckaert, F. Schuster, C *ACM Sigsac Conference on Computer and Communications Security*, pp. 243-255 (2015)

[10] W. Wu, S. Wu, L. Zhang, J. Zou, L. Dong. *International Conference on Information Security and Cryptology*, pp. 291-308 (2013)
[11] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici and I. Verbauwhede. IEEE Transactions on Computers, 62(10), pp.2041-2053 (2013)
[12] W. Wang, X. Wang, P. Du, Y. Tian, X. Zhang, Q. Hao, Z. Zhang and B. Xu. 22nd International Conference on Circuits, System, Communications and Computers, pp. 1-5 (2018)