Embedded System Confidentiality Protection by Cryptographic Engine Implemented with Composite Field Arithmetic

Weike Wang, *Xiang* Wang*, *Pei* Du, *Yuntong* Tian, *Xiaobing* Zhang, *Qiang* Hao, *Zhun* Zhang and *Bin* Xu

School of Electronic and Information Engineering, Beihang University, Beijing 100191, China

Abstract. Embedded systems are subjecting to various kinds of security threats. Some malicious attacks exploit valid code gadgets to launch destructive actions or to reveal critical details. Some previous memory encryption strategies aiming at this issue suffer from unacceptable performance overhead and resource consumption. This paper proposes a hardware based confidentiality protection method to secure the code and data stored and transferred in embedded systems. This method takes advantage of the I/D-cache structure to reduce the frequency of the cryptographic encryption and decryption processing. We implement the AES engine with composite field arithmetic to reduce the cost of hardware implementation. The proposed architecture is implemented on EP2C70 FPGA chip with OpenRisc 1200 based SoC. The experiment results show that the AES engine is required to work only in the case of I/D-cache miss and the hardware implementation overhead can save 53.24% and 13.39% for the AES engine and SoC respectively.

1 Introduction

The rapid development of the Internet of Things technology brings the flourish of the embedded systems. The performance of the embedded system can no longer be treated as the unique consideration for the embedded system and cyber-physical systems. Especially in some applications that are sensitive to information security, such as financial terminals, sensors, communication nodes and military devices, the security issues are even worth more of our attention than the speed of processing and operation. While the embedded systems expose themselves to the external networks and devices, they become more vulnerable to the security attacks. Most of the malicious attacks can lead to the deviation execution of the program or to reveal user's critical information [1]. Assuring the confidentiality of embedded systems can cope with these problems effectively.

However, there are some trade-offs in the design of embedded system confidentiality. The main challenge is the limited system resource constraints and the real-time requirements [2]. Embedded systems can't spare abundant resources for system reliability. In this paper, we address this question by providing a hardware based confidentiality protection method to secure the code and data stored in the external memory off the processor chip. We use a hardware AES engine integrated between the system bus and processor to implement the memory encryption and decryption operation. This scheme works with the processor instruction and data cache, which provides performance enhancements by reducing the number of encryption and decryption operations. In order to reduce the implementation cost of AES hardware, we use composite field arithmetic to optimize the hardware implementation of AES to reduce the area of the hardware required for the confidentiality protection.

The rest of this paper is organized as follows. In Section 2, we discuss some related works in the field of confidentiality protection. In Section 3, we present the proposed confidentiality protection scheme in details. Section 4 presents our experimental results and gives some analysis. Finally, we get a conclusion in Section 5.

2 Related works

Various works to address the embedded system security issue and protect system confidentiality have been published. Most of them put the emphasis on the static and dynamic analysis of the program, or the security of the cryptographic algorithm. These works are implemented in different ways, but almost all of them have their own limitations.

Some well know desktop security solutions have large system requirements and significant performance overhead, thus not suit to port to embedded systems. Runtime software monitoring and binary instrumentation techniques have been proposed since it was published by N. Oh [3]. But these solutions inevitably increase code sizes, bring performance overhead, or are themselves vulnerable to corruption. Wang [4] proposed a hardware assisted module to support program execution security. They divided program code into basic blocks by offline profiling and checking the code integrity at runtime. However, they may suffer from code reuse attacks (CRA) since the lack of sufficient security protections for the return-oriented programming (ROP) [5] or jump-oriented programming (JOP) attacks [6]. Lee proposed a

* Corresponding author: wxiang@buaa.edu.cn

hardware CRA monitor in an ARM-based SoC to detect ROP and JOP attacks through a system debug interface to trace outcomes [7]. However, these monitor measures [8] can only monitor such attacks, but can't provide effective defense strategies.

Memory encryption is an effective method to avoid CRA. CRA attacks exploit valid code gadgets to launch destructive actions or to reveal critical details. The encrypted code and data make the adversaries hard to understand the real implication of the code to perform CRA. Gueron [9] proposed constructions for memory protection using coprocessor and provided instruction set architecture support. However, this lacks generality and introduces unexpected vulnerability of its own. A security extension method for confidentiality has proposed by Rogers [10]. They encrypted the external code, data and private keys using AES, but this increase the cost of storage. In Wang's work [11], they implement an AES core in the SoC without considering any improvement of the feeding efficiency of the bus interface. Yang [12] shows solicitude for this issue, and proposed a way to improve AES core performance together with an advanced ASBUS protocol. They implemented the DMA and memory combined with AES engine in FPGA. However, the proposed architecture is connected to the memory controller. Similarly, multiple hard drivers and non-volatile memory (NVM) manufactures offers some self-encryption derives that the full disk and memory chip is encrypted entirely by the hard drive controllers [13]. If there are much external memory devices or memory chips in the system, there needs much more instance of the proposed module. This paper is to address the problems in previous works in system confidentiality preserving with low performance overhead and small footprints.

3 Embedded systems confidentiality protection method

In this section, we give some details in the proposed architecture used for embedded system confidentiality protection. We define the regions of the SoC chip as the trusted zone. The threat models come from malicious attacks outside the SoC chip. The devices, components, and wires off the chip are all assumed as incredible. Attacks such as code injection, data modification, and some ROP/JOP based advanced attacks can be performed by obtaining the data and code stored outside the trusted regions. To protect the confidentiality of embedded systems, there need to ensure that the out of chip data are encrypted and can't be cracked easily.

A hardware cryptographic engine is implemented in the SoC to achieve rapid encryption and decryption of the external memories. We employ AES as the cryptographic algorithm and implemented in hardware. The implementation and optimization of the AES engine will be illustrated in Section 4. The architecture details for the confidentiality protection method are shown in Fig. 1. In this microarchitecture, the plaintext is the code and data processed and transferred on the pipeline while the cipher is the masked data stored in the external memories. The private key is fixed in the SoC chip and consisted with the key used in the offline profiling stage following compilation. If the SoC is implemented on FPGA, the internal fixed private key can be modified during each configuration process.

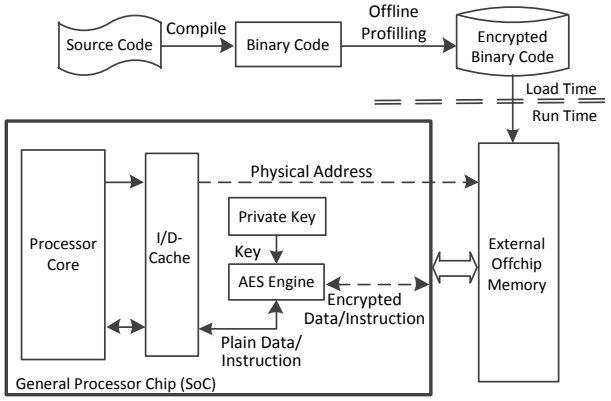

Fig. 1. Architecture details for the confidentiality protection method

Embedded system needs optimization to reduce the performance losses after adding a cryptographic engine. The AES engine is connected with the I/D-cache in this work to reduce the operating frequency. When the processor pipeline needs to fetch instruction and data, it may first access cache to inquire the plain code and data which has been decrypted or generated. The decryption engine only works in the case of cache miss. Similarly, the encryption engine works in the processor write back stage. A timing diagram of a read operation for the proposed architecture is shown as an example in Fig. 2. In case of the cache hit, a time interval of cache access is required merely. In high performance embedded processor, cache access time is usually within 2 clock cycles. If it works in the case of cache miss, the intervals may include the external memory access time and the decryption time. This will greatly increase the time of data fetch.

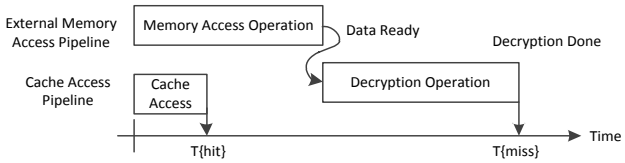

Fig. 2. The timing diagram of a read operation for the proposed architecture

In some previous works, the cryptographic modules are attached to the memory controller inside the SoC chip. This is inappropriate in some multiple external memory applications. Fig. 3 shows the architecture details of the proposed SoC in a multiple memory application. There are five memory devices in this system, and four of them are external memory chips. If the cryptographic hardware is connected to the controller, at least four AES engines need to be instantiated, which will cause a huge hardware waste. We put the AES engine between the processor and the system bus, and all external access through bus will be processed by this

engine. In this way, peripheral data transferred through the bus is also encrypted. We can avoid this issue by setting valid addresses inside the engine.

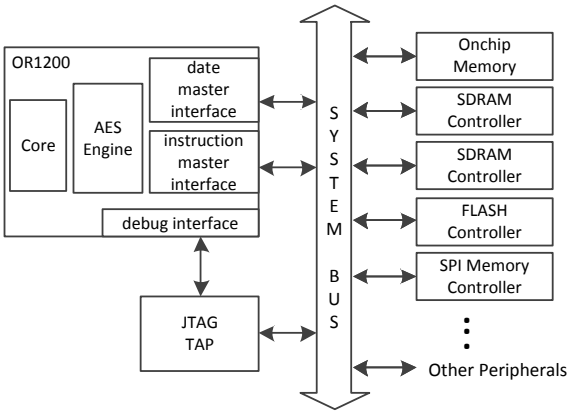

Fig. 3. Architecture details of the proposed SoC

4 AES Engine with Composite Field Arithmetic

AES is symmetric block cipher that processes on 128-bit data for 10 rounds, in form of a 4 × 4 matrix. In encryption, each round except the last one, four independent transformations including nonlinear byte substitution using S-box, shifting rows of the state array, mixing data of each column in the state array, and adding the round key are performed. In the last round, the final state will be taped out after the mix column operation. The instructions and data fetched by the pipeline are not stored sequentially in the external memories. Thus, the AES engine is implemented in electronic mode (ECB). Similarly, we have the decryption flow. In this section, we take the encryption process to illustrate the implementation of AES engine with composite field arithmetic.

AES process 8-bit state in finite field arithmetic on $GF(2^8)$ elements. The most common approach for the four steps is using lookup tables initialed by the aimed results. This may cause a lot of waste in hardware resources due to the storage units have a large area. In the mathematical theory, the finite field of $GF(2^8)$ can be built iteratively from $GF(2^4)$ using irreducible polynomials:

$$GF(2) \rightarrow GF(2^4) \quad : \quad P(x)=x^2+x+1 \tag{1}$$

$$GF(2^4) \rightarrow GF(2^8) \quad : \quad Q(x)=y^4+y+\beta \tag{2}$$

Where x is an element in GF(2), y and β are elements in $GF(2^4)$. In this work, we implement AES using $\beta = \{1100\}_2$.

The byte substitution transformation in encryption process can be divided into two steps, the modular inversion and affine transition over $GF(2^8)$. In the modular inversion stage, the state in $GF(2^8)$ is first mapped in $GF(2^4)$ using an isomorphic mapping matrix T. Then the inversion in $GF(2^8)$ is reduced to be inversion operation in $GF(2^4)$. After the reduced inversion operation, an inverse isomorphic mapping is performed. At last, an affine transition matrix is used in the affine linear function. The structure of the byte substation module in AES encryption process is shown in Fig. 4. The modular inversion operation in $GF(2^8)$ is decomposed as 3 steps linear operation and one step reduced modular inversion. The byte substation module is the most frequently used module in AES. In a round pipelined encryption implantation, there needs 20 byte substation modules, 16 for the byte substation stage and 4 for the mix column stage. The mix column transformation, shift row transformation, and round key addition involve some multiple and XOR computation in $GF(2^8)$, this is easy to implement in hardware as this operation can be mapped to gate elements and straight wires in circuits.

Table 1. The configuration of the embedded system

Content	Details
CPU	OR1200 @ 100MHz with 8K I/D-$
SRAM	2 Mbyte (IS61LS51236A)
SDRAM	32 Mbyte (IS42S16160B) * 2
FLASH	8 Mbyte (S29GL064A90)
AES engine	128 bit/ 10 round
GPIO	64-bit general purpose

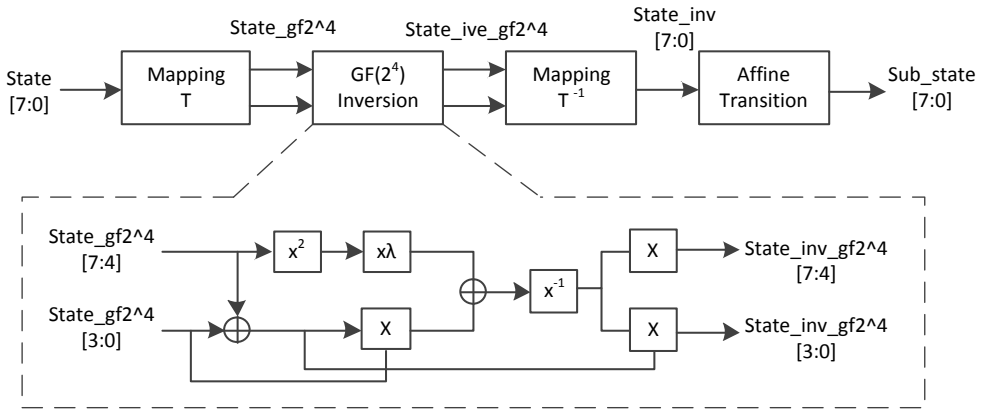

Fig. 1. Structure of the byte substation module in AES encryption process

Table 2. FPGA resource used for SoC

Resource Utilization		LUT Based	This Work	Saves
SoC	Logic Cells	13,792	11,118	19.39%
	Registers	4,379	4,379	-
	Memory Bits	556,416	556,416	-
AES	Logic Cells	4,940	2,310	53.24%
	Registers	530	530	-
	Memory Bits	384	384	-
S-Box	Logic Cells	208	67	67.79%

5 Experiment and results

In previous section, we demonstrated a hardware supported confidentiality protection method for embedded systems. In order to verify the effectiveness of this method, we implement the proposed architecture on the Altera EP2C70 FPGA platform. The processor used in this work is OR1200 which is a 32-bit scalar RISC core with Harvard microarchitecture. The configuration of the embedded system is listed in Table 1. In our experiment, the processor is configured with 8KB instruction cache and 8KB data cache. The processor core works at 100MHz, fully synchronized with the AES engine.

In the configuration of the embedded system, four external memory chips are employed. This is used to test whether the proposed AES engine and the micro architecture can work effectively in the encryption and decryption of multiple external memories. We selected various scales of benchmarks from Mibench suite to generate realistic workloads. The experiment shows that one AES engine can be used to encrypt and decrypt multiple external memories, when the AES hardware module is placed between the system bus and the processor core. This reduces the number of instances of the AES core and reduces the area cost of the embedded system. However, the performance overhead induced by the proposed confidentiality mechanism is not increased compared to the works attach the AES engines to the memory controllers.

To evaluate the hardware resource overhead of the AES engine proposed with composite field arithmetic, we implement AES engine both in LUT based method and composite field arithmetic based method. The FPGA resource used in this work is listed in Table 2. In order to get objective data, we implement the s-box in the fully combinational logic circuit. A state substation operation is completed in one clock cycle. The result shows that the composite field arithmetic based s-box can save 67.79% of the hardware resource compared to the LUT based work. Besides, the AES engine and SoC can save 53.29% and 19.39% respectively. Additionally, the area of the AES engine is only 20.78% of the combination logic area of the SoC, even though the OR1200 is a relatively smaller core compared with typical embedded processors, such as ARM, MIPS, and PowerPC. This is an acceptable result in the area cost of embedded systems.

6 Conclusions

This paper proposes a hardware based confidentiality protection method for embedded systems. All the data and program code stored off the SoC chip is encrypted using AES algorithm to secure the critical information. We implement the hardware AES encryption and decryption engine between the data/instruction bus and the processor to realize the engine sharing of multiple external memory chips. Moreover, this microarchitecture takes advantage of the I/D-cache structure to reduce the frequency of the AES encryption and decryption processing. In order to further reduce the cost of area and power, we optimize the hardware implantation of the AES engine, especially for the substation box with composite field arithmetic. The experiment results show that the proposed method can perform effective confidentiality protection for the embedded systems with lower performance costs and smaller hardware footprints.

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).

References

1. D.N. Serpanos, A.G. Voyiatzis: Security challenges in embedded systems. ACM Trans. on Embed. Comp. Sys. **12(1s)**, 66:1-10 (2013).
2. A. Aminifar, P. Eles, Z. Peng: Optimization of message encryption for real-time applica-tions in embedded systems. IEEE Trans. On Comp. **67(5)**, 748-754 (2018).
3. N. Oh, P. Shirbvani, E. McCluskey: Control-flow checking by software signatures. IEEE Trans. on Reli. **51(2)**, 111-122 (2002).
4. X. Wang, S. Pang, W. Wang, et al.: Hardware-assisted system for program execution se-curity of SoC. In: ITA 2016, pp. 1-6. EDP Science (2016).
5. M. Prandini, M. Ramilli: Return-oriented programming. IEEE Secu. and Priv. **10(6)**, 84-87 (2012).
6. T. Bletsch, X. Jiang, Freeh, et al.: Jump-oriented programming: A new class of code-reuse attack. In: ASIACCS 2011, pp. 30–40. ACM (2011).

7. Y. Lee, J. Lee., I. Heo and et al.: Integration of ROP/JOP monitoring IPs in an ARM-based Soc. In: IEEE DATE 2016, pp. 331-336. IEEE (2016).
8. S. Das, W. Zhang, Y. Liu: A fine-grained control flow integrity approach against runtime memory attacks for embedded systems. IEEE Trans. on VLSI **24(11)**, 3193-3207 (2016).
9. S. Gueron: Attacks on encrypted memory and constructions for memory protection. In: FGTC 2016, pp.1-3. IEEE (2016).
10. A. Rogers, A. Milenkovic: Security extensions for integrity and confidentiality in embedded processors. Micropro. and Microsys. **33**, 398-414 (2009).
11. X. Wang, B. Xu, W. Wang and et al.: A Novel Security Validation in Embedded System. In: IEEE ICSESS 2017, pp. 632-635. IEEE (2017).
12. X. Yang, W. Wen, M. Fan: Improving AES core performance via an advanced ASBUS protocol. ACM J. on Emer. Tech. in Comp. Syst. **14(1)**, 6: 1-23 (2017).
13. M. Henson, S. Taylor: Memory Encryption: A survey of Existing Techniques. ACM Comp. Surv. **46(4)**, 53:1-26 (2014).