International Journal of Security and Its Applications
Vol. 9, No. 4 (2015), pp. 117-124
http://dx.doi.org/10.14257/ijsia.2015.9.4.12

Low-Cost Round Encryption Method for Embedded System

Qi Yue, Luo Xinqiang and Wan Yadong

School of Computer and Communication Engineering
University of Science and Technology Beijing, China
qiyuee@ustb.edu.cn

Abstract

Advanced Encryption Standard (AES) is used in many security systems for its high security. However, the high computation complexity of the round encryption of AES makes AES hard to be performed on many embedded systems with constrained resource. This paper proposes a novel round encryption method based on a 512-Byte lookup table. The experiments results show that, the AES implementation utilizing the round encryption method proposed reaches an encryption performance comparable to that of the widely used implementation with 4 1-KB lookup tables, while consumes much less storage overhead.

Keywords: *Round encryption, AES, Lookup table, Low-cost, Embedded system*

1. Introduction

Advanced Encryption Standard (AES)[1] is a symmetric key cipher algorithm widely used in a great deal of security systems[2] and communication systems[3-4] for its high security and flexibility. However, the high computing complexity of AES makes AES hard to be performed on many resource-constrained systems such as embedded systems of which the power are supplied by batteries.

AES is made up of 10/12/14 rounds of round encryptions corresponding to 128/196/256 bits of key lengths separately. The majority computation of AES comes from the multiple rounds of round encryptions containing a great deal of Galois Filed GF(256) multiplications which are not supported by most of the processors widely used today and must be realized by number of instructions.

The AES hardware accelerators[5-6] can achieve high encryption performance. However, most of the AES accelerators are targeted on high performance encryption but not the low-cost embedded systems. Besides, the additional energy consumed by the hardware accelerators is also a heavy burden to the batteries. The instruction set extensions[7-8] are also can improve AES encryption performance, but these methods limit the choice of processors, and are not suitable to embedded systems applied in various fields. Software optimizations[9-11] focus on the AES software implementation itself, and is independent of hardware. The fastest AES software implementation is proposed by Gladman[10]. Named as 4-T, the implementation utilizes 4 1-KB lookup tables (LUT), and converts the complex round encryption into just 16 times looking up LUTs and 16 times XOR operations. However, since it contains 4 1-KB LUTs, makes it not suitable to the systems with constrained storage resource.

According to this problem, this work proposes a novel round encryption method based on a 512B LUT for the resource-constrained embedded systems. Since most of the encryption mode[12] widely used today involve only the encryption proceed of the cipher algorithm, this work focus on the optimization of the round encryption of the encryption proceed only.

ISSN: 1738-9976 IJSIA

The reminder of this paper is organized as follow. In section II we give a brief description of AES round encryption. We describe details of our 1-T design in section III and analyze its overhead in section IV. A typical application of secure data transmission, CCM mechanism, is discussed in section V. Conclusion is drawn in section Ⅵ.

2. AES Round Encryption

AES round encryption works on the 4×4 state array of plaintext (A), and contains: AddRoundKey, adding round key (KR) and A; SubBytes, replacing each byte of A with another according to a lookup table S-box; ShiftRows, shifting the last three rows of A cyclically a certain number of steps; and MixColumns, operating on the columns of A and combines the four bytes in each column by doing GF(28) multiplication with mixing array $C = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$.

Assuming $A = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$ and $KR = \begin{bmatrix} kr_0 & kr_4 & kr_8 & kr_{12} \\ kr_1 & kr_5 & kr_9 & kr_{13} \\ kr_2 & kr_6 & kr_{10} & kr_{14} \\ kr_3 & kr_7 & kr_{11} & kr_{15} \end{bmatrix}$, the round encryption can be expressed as follow:

$$fr(A,KR) = C \otimes \begin{bmatrix} S(a_0) & S(a_4) & S(a_8) & S(a_{12}) \\ S(a_5) & S(a_9) & S(a_{13}) & S(a_1) \\ S(a_{10}) & S(a_{14}) & S(a_2) & S(a_6) \\ S(a_{15}) & S(a_3) & S(a_7) & S(a_{11}) \end{bmatrix} \oplus KR \quad (1)$$

Where $S(a)$ represents the result of SubBytes; $\otimes$ represents GF(256) multiplication; and $\oplus$ represents XOR.

3. Single LUT based Round Encryption

3.1 Low-Storage LUT

The main idea of 4-T is extracting $C \otimes \begin{bmatrix} S(a_0) & S(a_4) & S(a_8) & S(a_{12}) \\ S(a_5) & S(a_9) & S(a_{13}) & S(a_1) \\ S(a_{10}) & S(a_{14}) & S(a_2) & S(a_6) \\ S(a_{15}) & S(a_3) & S(a_7) & S(a_{11}) \end{bmatrix}$ of (1), so that each $S(a)$ is only multiplied with a certain column of C, and then converting these multiplications into looking up tables storing the products.

The structures of the 4 1-KB LUTs of 4-T are show as follow:

TS2113: storing $[2\ 1\ 1\ 3]^T \otimes S(a)$ (representing as $S_{2113}(a)$);

TS3211: storing $[3\ 2\ 1\ 1]^T \otimes S(a)$ (representing as $S_{3211}(a)$);

TS1321: storing $[1\ 3\ 2\ 1]^T \otimes S(a)$ (representing as $S_{1321}(a)$);

TS1132: storing $[1\ 1\ 3\ 2]^T \otimes S(a)$ (representing as $S_{1132}(a)$).

Observing the four lookup tables of 4-T, any three of them can be obtained by shifting the other one cyclically. For example, TS3211 can be obtained by left shifting TS2113 one byte cyclically. Thus, only one table needs to be stored while the other three ones can be obtained by shifting the stored one. Then the storage of LUT drops to 1 KB, requiring some simple additional shift operations to obtain the other 3 results.

Since each item of any of the 4 LUTs of 4-T contains 2 $1 \otimes S(a)$, 1 $2 \otimes S(a)$ and 1 $3 \otimes S(a)$, only 3 tables TS1, TS2 and TS3 are needed to store $1 \otimes S(a)$, $2 \otimes S(a)$ and $3 \otimes S(a)$ separately. Moreover, according to GF(256) multiplication, $3 \otimes S(a)$ can be obtained by simple computation: $3 \otimes S(a) = 1 \otimes S(a) \oplus 2 \otimes S(a)$, so the round encryption just

needs to look up TS1 and TS2, then all the GF(2^8) multiplication results for MixColumns can be obtained easily. In this way, the storage of lookup table drops to 512 Bytes. But additional operations are needed to combine the byte-wise results into the word-wise results $S_{2113}(a)$, $S_{3211}(a)$, $S_{1321}(a)$ and $S_{1132}(a)$.

3.2 Operation Optimization

In order to reduce the additional operations, a novel LUT is proposed in this part by combining the two LUTs TS1 and TS2 mentioned above.

Usually, the combination of word-wise results requires moving the byte-wise results into the word-wise results one by one by SHIFT and OR operations. If multiple byte-results can be moved into the word-wise result in one time, some operations can be saved. By observing the four word-wise results, as Fig.1 and Fig.2 showing, some $S_1(a)$ and $S_2(a)$ connections exist.

As Fig.1 showing, 3 $S_1(a)S_2(a)$ connections are existing in the four word-wise results, and no $S_2(a)S_1(a)$ connection existing. But since the $S_3(a)$ is gained by $3\otimes S(a)=1\otimes S(a)\oplus 2\otimes S(a)$, it forms a $S_2(a)S_1(a)$ with the $S_1(a)$ of its right side and another one with the $S_2(a)$ of its left side. So there are 6 $S_2(a)S_1(a)$ connections existing as Fig.2 showing. That means $S_2(a)S_1(a)$ can save more operations than $S_1(a)S_2(a)$ does.

By combining TS1 and TS2 into one 512-Byte LUT TS21 which stores half-word-wise result $S_2(a)S_1(a)$ (expressing as $S_{21}(a)$), the $S_2(a)S_1(a)$ can be gained by looking up TS21.

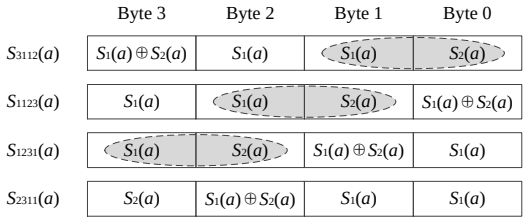


Figure 1. $S_1(a)S_2(a)$ Contained in Word-wise Results

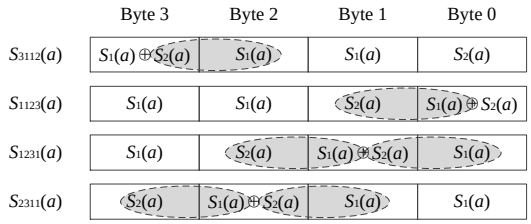


Figure 2. $S_2(a)S_1(a)$ Contained in Word-wise Results

Besides, by combining LUT, only one memory access is needed for each element of the state. Since memory access instructions consume more cycles than other arithmetical instructions for most processor we using today, the more memory access instructions are saving, the more cycles are saving, reducing the more encryption time.

However, the byte-wise results $S_1(a)$ and $S_2(a)$ are still required for the combination of word-wise results, so additional operations for splitting the byte-wise results from $S_{21}(a)$:

$$S_2(a) = S_{21}(a) >> 8 \quad (2)$$

$$S_1(a) = S_{21}(a) \,\&\, 0xff \quad (3)$$

$$S_1(a) << 24 = S_{21}(a) << 24 \tag{4}$$

The round encryption of the first column can be expressed as follow:

$$\begin{aligned} &b_3 b_2 b_1 b_0{}^T = \\ &((S_{21}(a_0) << 24) \oplus (S_{21}(a_0) << 16) \mid ((S_{21}(a_0) \,\&\, 0xff) << 8) \mid (S_{21}(a_0) >> 8)) \oplus \\ &((S_{21}(a_5) << 24) \mid ((S_{21}(a_5) \,\&\, 0xff) << 16) \mid (S_{21}(a_5) \oplus (S_{21}(a_5) >> 8))) \oplus \\ &((S_{21}(a_{10}) << 24) \mid (S_{21}(a_{10}) << 8) \oplus S_{21}(a_{10})) \oplus \\ &((S_{21}(a_{15}) << 16) \oplus (S_{21}(a_{15}) << 8) \mid (S_{21}(a_{15}) \,\&\, 0xff)) \oplus \\ &kr_3 kr_2 kr_1 kr_0{}^T \end{aligned} \tag{5}$$

Table 1. Operation of 1-T Round Encryption

Operation	Count
Load	16
XOR	32
LSL	36
OR	24
LSR	12
AND	8

The other three columns can be express in the same way. This method is named as 1-T. Table 1 shows the operations included in the round encryption of 1-T. We can see that more operations are needed for 1-T than 4-T which requires only 16 Load operations and 16 XOR operations. But the advantage is that the memory requirement for the LUT drops to 512-Byte from 4-KB.

4. Overhead Analysis

This section will estimate the overhead of the AES implementation based on 1-T round encryption. The estimation is based on ARM CortexM3 CPU STM323F103RE working at 72MHz. For an objective estimation, the implementations of computing GF(256) multiplication directly (GF), 4-T and the implementation of hardware AES (HW) based on radio AT86RF231ZU are involved.

4.1 Storage

Fig.3 shows the storage overhead of the four AES implementations mentioned above. Since most of the computations of HW are performed on hardware, its code storage and data storage mainly come from the SPI communication and are the least of the four implementations. 1-T's round encryption is more complex than the other three ones, so its code consumed the most storage, but just 29.3% higher than the HW's. For data storage, since 4-T contains 4 1KB LUTs, its data storage reaches 4540 Bytes and is the highest; GF just contains a 256 Bytes S-box, so its data storage is only 444 Bytes; 1-T stores a 512 Bytes LUT, so its data storage is just 57.6% higher than GF's and only 15.4% of 4-T's.

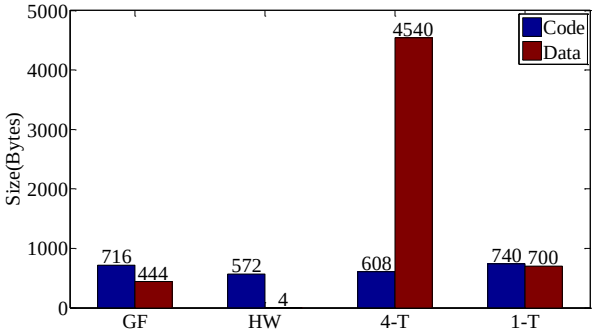


Figure 3. Storage Overhead of 4 Implementations

4.2 Time

Fig.4 shows the time taken by the four AES implementations to encrypt a plaintext block. Although HW performs AES encryption on hardware which is faster than software, the additional time taken by SPI communication makes HW's encryption time much longer than 4-T's and even 1-T's. Since GF computes the time-consuming GF(28) multiplications directly, its encryption time is two orders of magnitude longer than the other three ones'. Benefit from the 4 large LUTs, 4-T has a simple round encryption function and consumes the shortest time for encryption. Since 1-T needs to split and combine byte-wise results, its encryption time is 43.5% longer than 4-T's.

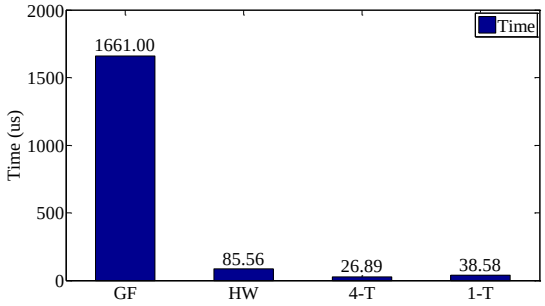


Figure 4. Time Overhead of the 4 Implementations

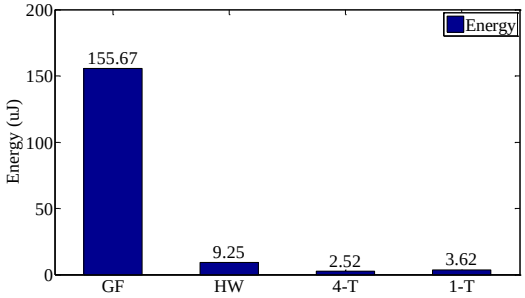


Figure 5. Energy Overhead of the 4 Implementations

4.3 Energy

The energy cost by the three software implementations to encrypt a plaintext block can be computed as follow.

$$E_{sw} = I_{CPU} * U_{CPU} * t_{CPU} \tag{6}$$

Where I_{CPU} is the current of only CPU working; U_{CPU} is the supply voltage; and t_{CPU} is the execution time of software.

The energy overhead of HW is divided into two parts: one is the energy consumed by CPU and SPI when SPI is working; and the other is energy consumed by CPU and AES hardware when CPU is waiting and AES hardware is working. The energy consumed by HW to encrypt a plaintext block can be computed as follow:

$$E_{hw} = I_{CPU+SPI} * U_{CPU} * t_{CPU+SPI} + I_{CPU+HW} * U_{CPU} * t_{CPU+HW} \tag{7}$$

Where $I_{CPU+SPI}$ and $t_{CPU+SPI}$ are the current and time separately when both CPU and SPI are working; I_{CPU+HW} and t_{CPU+HW} are the current and time separately when both CPU and AES hardware are working.

The currents of different modes are shown in Table 2. According to those currents, the energy consumed by the four AES implementations to encrypt a plaintext block is shown in Fig.5. As shown in the figure, since the energy consumed by the software implementations is proportional to their execution time, 1-T's energy overhead is a little higher than 4-T's and much lower than GF's as their time overhead. Besides the energy consumed by CPU, the additional energy consumed by AES and SPI makes HW's energy overhead 3.67 times of 4-T's and 2.56 times of 1-T's.

Table 2. Currents of Different Modes

	CPU	CPU+SPI	CPU+HW
Current(mA)	28.4	33.8	28.6

5. Typical Application

In the current mainstream industrial wireless network protocols (IEEE 802.15.4 [4], ISA100 [13], WirelessHART[14], WIA-PA[15]),the secure data transmission has been defined. As a widespread secure data transmission mechanism in wireless networks, CCM[12] is adopted by these protocols as an only or optional secure transmission mechanism in MAC layer. The core encryption method used in CCM is 128-bit AES.

This section will apply the four implementations of AES mentioned above to CCM, and estimate the different overhead of CCM. All the three software implementations are performed on CPU, while the hardware auxiliary implementation is performed on the AES module in the radio chip.

5.1 CCM Computation

CCM supports both message encryption service and message integration check service. Message encryption uses Counter Mode (CTR) to encrypt message and message integration check computes message encryption code (MIC) in Cipher Block Chaining mode (CBC). Fig. 6 shows the AES computation times taken by CTR (encryption/decryption) and CBC(MIC computation/checking) in one CCM computation at different payload length. As MIC computation involves the whole packet while encryption only involves the payload field, the computation times of AES for CBC are more than that for CTR. It can be seen from Fig.6 that the computation times of AES increase along with the length increasing of packet.

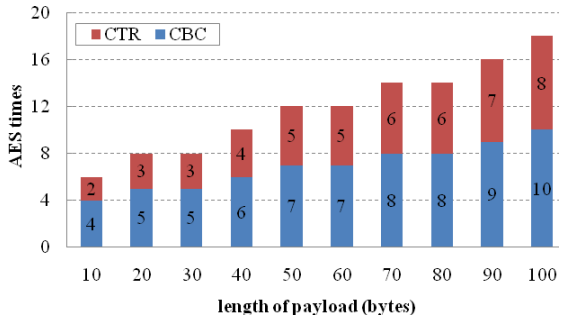


Figure 6. AES Computation Times in One CCM Computation

5.2 CCM Storage Analysis

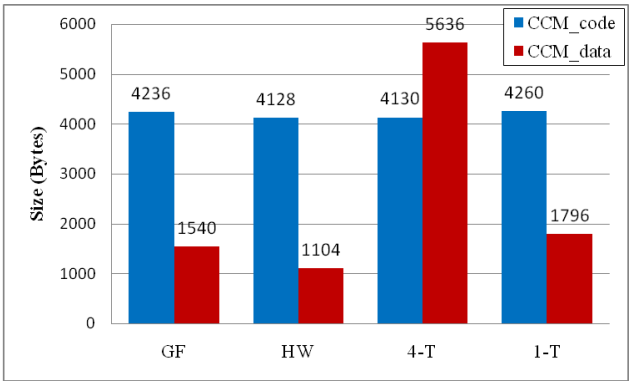


Figure 7. Storage Comparison of CCM

The storage overheads of CCM in the four implementations are illustrated in Fig. 7. According to the figure, whether the AES is implemented in three software implementations or the hardware one, the occupancy of CCM_code storage are the same, while the storage of CCM_data are different obviously. It is because that the AES overhead is difference.

According to Fig. 3 and Fig. 7, the data storage of AES which is implemented in 4-T is 4540 bytes, which is far more than the data storage of entire CCM which is implemented in other three methods, even more than the code storage of entire CCM. For the data storage of CCM, 1-T implementation is 1796, which is 62.6% higher than the HW's, and just16.6% higher than GF's and only 31.9% of 4-T's.

5.3 CCM Performance Analysis

Fig. 8 shows the proportion of AES computation time in one CCM computation. As the figure shows, in the GF implementation and HW implementation, the AES computation takes the most proportion of CCM computation time and the proportion is increased with the payload length. This is because that: (1)the AES computation time is long; (2) the longer the payload length is, the more times of AES it takes, causing the proportion of AES computation time increased.

According to Fig. 8, the AES computation proportion in the 4-T and 1-T implementation is very lower than other two's. Benefit from the 4-T and 1-T's short AES encryption time and consume the little AES occupancy. 4-T and 1-T's AES computation proportion only 6%-10% of GF's, and 9%-15% of HW.

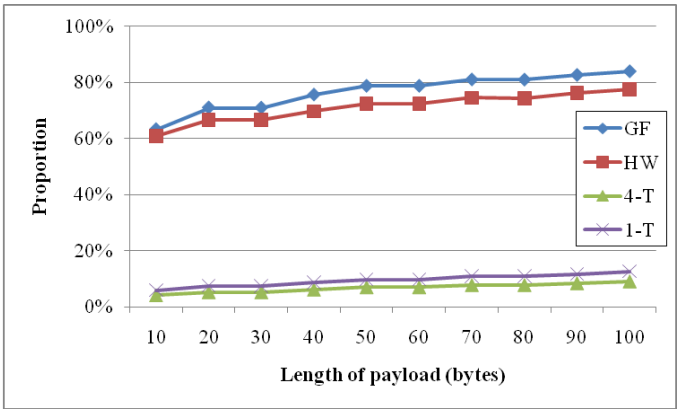


Figure 8. Proportion of AES Computation Time in One CCM Computation

6. Conclusion

For the resource-constrained embedded systems, this work proposes a low-cost round encryption method 1-T for AES encryption, based on single 512-Byte LUT. Compare with the fastest AES software implementation based on 4 1-KB LUTs and the implementation based on hardware accelerator, the implementation based on 1-T shows great advantage in the storage respect, and maintains a considerable encryption performance as 4-T.

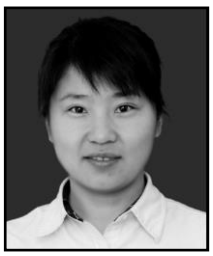
Acknowledgement

This work is supported by the National High Technology Research and Development Program of China (No. 2011AA040101).

References

[1] J. Daemen and V. Rijmen, "AES proposal: Rijndael", First Advanced Encryption Standard (AES) Conference, **(1998)**.
[2] J. Viega, M. Messier and P. Chandra, "Network Security with OpenSSL: Cryptography for Secure Communications", O'Reilly Media, Inc., **(2002)**.
[3] IEEE Standards Association, 802.11-2012-IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, **(2012)**.
[4] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LRWPANs), IEEE Computer Society Std. 802.15.4, **(2007)**.
[5] K. Rahimunnisa, P. Karthigaikumar and J. Kirubavathy, "A 0.13-μm implementation of 5 Gb/s and 3-mW folded parallel architecture for AES algorithm", International Journal of Electronics, vol. 1-12, **(2013)**.
[6] S. Morioka and A. Satoh, "A 10-Gbps full-AES crypto design with a twisted BDD S-Box architecture", IEEE Transactions on VLSI Systems, vol. 12, no. 7, **(2004)**, pp. 686-691.
[7] L. Xu, Secure the Enterprise with Intel® AES-NI: White Paper. http://www.intel.cn/content/www/cn/zh/enterprise-security/enterprise-security-aes-ni-white-paper.html.
[8] R. B. Lee and Y. Y. Chen, "Processor accelerator for AES", IEEE 8th Symposium on Application Specific Processors, **(2010)**; Anaheim, United states.
[9] G. Bertoni, L. Breveglieri and P. Fragneto, "Efficient software implementation of AES on 32-bit platforms", Cryptographic Hardware and Embedded Systems-CHES, **(2003)**, pp. 159-171.
[10] B. Gladman, "A Specification for Rijndael, the AES Algorithm", Available at http://fp.gladman.plus.com, **(2002)**.
[11] K. Atasu, L. Breveglieri and M. Macchetti, "Efficient AES implementations for ARM based platforms", Proceedings of the ACM symposium on Applied computing, ACM, **(2004)**; Nicosia, Cyprus.
[12] D. Whiting, N. Ferguson and R. Housley, Counter with cbc-mac(ccm), **(2003)**.
[13] An ISA Standard Wireless systems for industrial automation: Process control and related applications, ISA Std. ISA-100.11a, **(2009)**.
[14] HART Communication Foundation (HCF), Std. Available:http://www.hartcomm2.org/index.html.
[15] Industrial wireless networks WIA specification - Part 1:WIA System architecture and communication specification for process automation(WIA-PA), GB/T 26709.1, **(2011)**.

Authors



Qi Yue, female, born in 1975, instructor, her main research interests include micro-architecture and embedded system and wireless sensor network.

