Implementation of Advanced Encryption Standard for Cryptographic Applications using Arm 9 Processor on Linux Platform

Pradeep Karanje[1] Dr.Ravindra.E[2]
[1]Associate Professor [2]Professor and H.O.D
[1,2]Guru Nanak Dev Engineering College, Bidar.

Abstract— Now days, due to rapid development in technology, it is possible for us to have hi-tech secure products. Furthermore, the customers want their application to be run on kernel level. In this paper, we have implemented Advanced Encryption Algorithm for various cryptographic applications on Linux platform using ARM9 processor. The ARM processors are widely used in various embedded applications because of Low power and Low cost. We have used friendly ARM9 board for our development. Our work concentrates on ARM9 core widely used in the embedded industry. Most promising implementation choices for the common ARM9 Instruction Set Architecture (ISA) are identified, and a new implementation for the linear mixing layer is proposed. The performance improvement over current implementations is demonstrated by a case study on the Samsung Strong ARM9 S3C2440 Microprocessor.

Key words: AES algorithm, ARM 9, Kernel, Cipher text, Plain text

I. INTRODUCTION

Security of the data is important in many applications such as, military applications, examination papers encryption system. For this mechanism we are using AES algorithm which was proposed in the year 2001 by NIST. This AES gives the resistance from the known attacks using the encryption algorithm. In this paper we are taking theplain text as an input and giving it to the encryption algorithm which produces the cipher text output. The cipher text output is given as an input to the decryption algorithm which gives the text output which is same as the given input plain text.AES comprises three blocks of ciphers such as AES 128, 192,256 in which we are using AES 256 for encryption and decryption.We are processing the executive values on ARM 9 friendly processor and showing the results on the main screen of it. For execution we need to dump the hex file in the ARM 9 base.

II. AES MECHANISM

Key Expansion—round keys are derived from the cipher key using Rijndael's key schedule.
1) Initial Round
Add Round Key—each byte of the state is combined with the round key using bitwise xor.
Rounds
- Sub Bytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
- Shift Rows—a transposition step where each row of the state is shifted cyclically a certain number of steps.

2) Mix Columns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
3) 4Add Round Key
4) Final Round (no Mix Columns)
5) Sub Bytes

A. *Add round key:*

In the Add Round Key step, each byte of the state is combined with a byte of the round sub key using the XOR operation ($\oplus$).

In the Add Round Key step, the sub key is combined with the state. For each round, a sub key is derived from the main key using Rijndael's key schedule; each sub key is the same size as the state. The sub key is added by combining each byte of the state with the corresponding byte of the sub key using bitwise XOR.

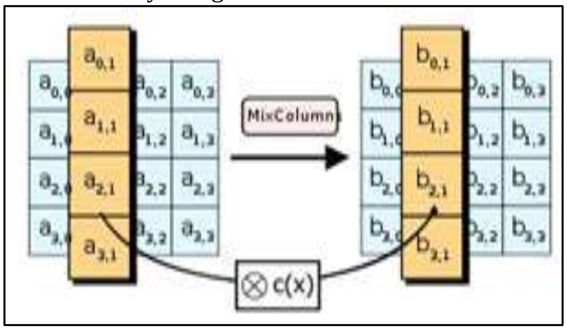
MixColumn
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$\otimes c(x)$

Fig. 1: Add round key operation

B. Sub—bytes:

In the Sub-Bytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S;bij = S(aij).
In the Sub-Bytes step, each byte in the state matrix is replaced with a Sub-Byte using an 8-bit substitution box, the Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.

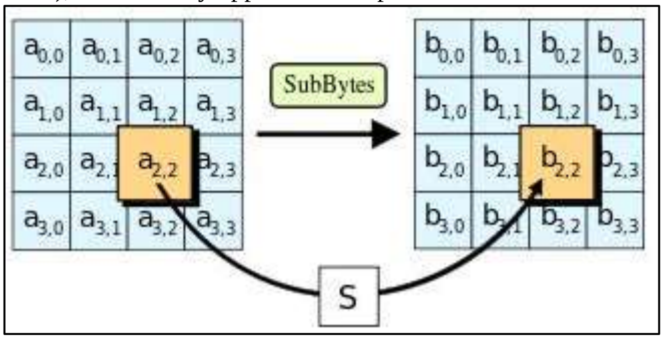
SubBytes
S
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Fig. 2: Sub bytes operation

C. Shift rows:

In the Shift Rows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.

The Shift Rows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row n is shifted left circular by n-1 bytes. In this way, each column of the output state of the Shift Rows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to make columns not linear independent if so, AES becomes four independent block ciphers.

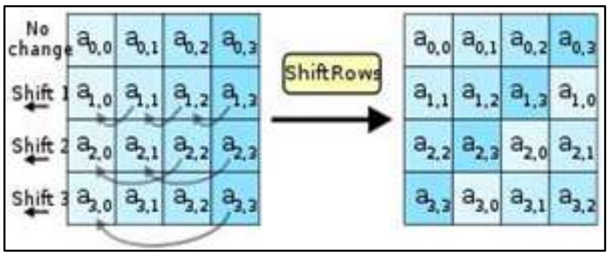
No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Fig. 3: Shift rows operation

D. Mix columns:

In the Mix Columns step, each column of the state is multiplied with a fixed polynomial c(x).In the Mix Columns step, the four bytes of each column of the state are combined using an invertible linear transformation. The Mix olumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with Shift Rows, Mix Columns provides diffusion in the cipher.During this operation, each column is multiplied by the known matrix that for the 128-bit key is:

The multiplication operation is defined as: multiplication by 1 means no change, multiplication by 2 means shifting to the left, and multiplication by 3 means shifting to the left and then performing xor with the initial unsuited value. After shifting, a conditional xor with 0x11B should be performed if the shifted value is larger than 0xFF.

In more general sense, each column is treated as a polynomial over GF(28) and is then multiplied modulo x4+1 with a fixed polynomial c(x) = 0x03 • x3 + x2 + x + 0x02. The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from GF(2)[x]. The Mix Columns step can also be viewed as a multiplication by a particular MDS matrix in a finite field. This process is described further in the article Rijndael mix columns.

III. ARM9 PROCESSOR

Computer processors designed and licensed by British company ARM Holdings. It was first developed in the 1980s by Acorn Computers Ltd to power their desktop machines and subsequently spun off as a separate company, now ARM Holdings. Globally as of 2013 it is the most widely used 32-bit instruction set architecture in terms of quantity produced. According to

ARM Holdings, in 2010 alone, producers of chips based on ARM architectures reported shipments of 6.1 billion ARM-based processors, representing 95% of smart phones, 35% of digital televisions and set-top boxes, and 10% of mobile computers.

As an IP core business, ARM Holdings itself does not manufacture its own electronic chips, but licenses its designs to other semiconductor manufacturers. ARM-based processors and systems on a chip include the Qualcomm Snapdragon, nVidia Tegra, Marvell Xscale and Texas Instruments OMAP, as well as

ARM's Cortex series and Apple System on Chips (used in its iPhones). The name was originally an acronym for Acorn RISC Machine and subsequently, after the name Acorn was dropped, Advanced RISC Machine.

With this design generation, ARM moved from a von Neumann architecture (Princeton architecture) to a Harvard architecture with separate instruction and data buses (and caches), significantly increasing its potential speed. Most silicon chips integrating these cores will package them as modified Harvard architecture chips, combining the two address buses on the other side of separated CPU caches and tightly coupled memories.
There are two subfamilies, implementing different ARM architecture versions.
Differences from ARM7 cores:

- Key improvements over ARM7 cores, enabled by spending more transistors, include:
- Decreased heat production and lower overheating risk.
- Clock frequency improvements. Shifting from a three stage pipeline to a five stage one lets the clock speed be approximately doubled, on the same silicon fabrication process.
- Cycle count improvements. Many unmodified ARM7 binaries were measured as taking about 30% fewer cycles to execute on ARM9 cores. Key improvements include
- Faster loads and stores; many instructions now cost just one cycle. This is helped by both the modified Harvard architecture (reducing bus and cache contention) and the new pipeline stages.

Exposing pipeline interlocks, enabling compiler optimizations to reduce blockage between stages. Additionally, some ARM9 cores incorporate "Enhanced DSP" instructions, such as a multiply-accumulate, to support more efficient implementations of digital signal processing algorithms.

Switching to a Harvard architecture entailed a non-unified cache, so that instruction fetches do not evict data (and vice versa). ARM9 cores have separate data and address bus signals, which chip designers use in various ways. In most cases they connect at least part of the address space in von Neumann style, used for both instructions and data, usually to an AHB interconnect connecting to a DRAM interface and an External Bus Interface usable with NOR flash memory. Such hybrids are no longer pure Harvard architecture processors.

IV. EXPERIMENTAL ANALYSIS

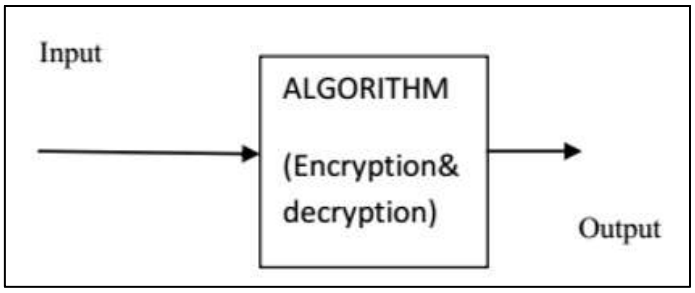

Fig. 5: Level1 Execution

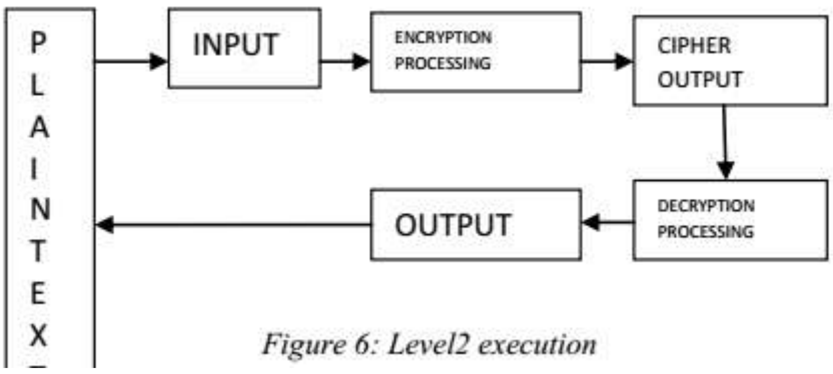

Figure 6: Level2 execution

V. IMPLEMENTATION

In our implementation phase, first we have to set up ARM9 tool chain for code compilation in Linux platform. The firmware has to be developed by using C Program. After the compilation, the hex file is mapped to our RAM9 board using UART communication. A simulation campaign has been carried out, in order to evaluate time performances. Initially we have tested the results of the AES optimized algorithm only in the case of a key size of 128 bits, but we also tested the proposed algorithm with 192 and 256 bits of key size. The results are quite the same as those obtained with 128 bits key size, only scaled by a constant factor due to the larger number of rounds required by these versions of the AES algorithm. The time performance gain of each round with respect to the standard AES algorithm remains the same in all the cases.

We have compiled it on X86 platform and sent the obtained output using RS232 to display it on ARM9 processor and Our code has been compiled and evaluated for ARM9 processor. The results of our implementation is as shown in the following figures

Fig. 7: Encryption and Decryption using AES

VI. Conclusion

In our work, the peculiarities of the common ARM processors suited for AES are identified and appropriately exploited. A new implementation for the encryption linear mixing layer is formulated, which enhances the performance of AES on all ARM cores. We have rewritten the basic transformations of the Rijndeal Cipher algorithm, using a transposed version of the so called State matrix. We have shown that this relevant structural modification leads to an improvement of time performances in decryption. As for encryption, the time performances of our version of AES and Gladman's are instead more or less the same. These results hold when a limited part of AES computation is carried out using look- up tables. Namely we have supposed that only the S – BOX operation is tabularized and stored in a look- up memory.

References

[1] NIST Specification for the advanced encryption standard (aes), fips pubs 197, November 26, 2001.
[2] J. Daemen and V. Rijmen AES Proposal: Rijndael, NIST AESProposal, Dr. Dobb's Journal Vol. 26 , No. 3, March2003.928
[3] Andrew N. Sloss, Dominic Symes, Chris Wright, "ARM System Developer's Guide Designing and Optimizing System Software"
[4] Charlie Kaufman, Radia Perlman, Mike Speciner, "Network Security" second Edition PHI -2003.
[5] B. Gladman. A Specification for Rijndael, the AES Algorithm. Available at http://fp.gladman.plus.com, May 2002
[6] Configuring ARM Caches. Application Note ARM DAI 0053B, ARM Limited, Feb 1998.
[7] Announcing the advanced encryption standard (AES). Federal Information Processing Standard FIPS 197, National Institute of Standards and Technology (NIST), Nov 2001.